The Teaching Portfolio of Emily Jacobson

The College Classroom

Fall 2011

Contents

Teaching Philosophy

My approach to teaching has been shaped by my interactions as both a student and a teacher: I see the teacher not only as a role model and instructor, but also a colleague and peer. I believe one of the most important roles of a teacher is to guide students towards knowledge. More than that, I want to challenge students to think critically about the material and encourage them to seek out new information on their own.

Computer Science, in particular, is an interesting field in which to teach because people often have a variety of preconceived notions about what Computer Science is and who can be a Computer Scientist. An important part of teaching is dispelling these myths and encouraging all students, especially non-traditional CS students, to explore and excel at Computer Science.

The classroom should be an interactive, collaborative environment. I believe in this type of environment for two reasons. First, class is more exciting for students when they are actively learning rather an passively absorbing knowledge. Second, I think it is both more effective and more efficient have students interact with each other and the material. When I lecture, I involve students in the learning process. This means asking questions and informally calling on students. Some students are more likely to volunteer answers; I want to involve these students while encouraging the more reluctant students to participate. In my experience, students who know their peers are more comfortable answering questions or putting forth their own. I believe time spent on introductions at the beginning can be beneficial throughout the semester. However, because not all students will feel comfortable participating in class discussions, I want to provide other opportunities for these students to interact with the material, such as online discussion boards or small group discussions.

By collaborating to understand new material, students can more effectively grasp new concepts. When possible, I want students to work in groups to solve problems and process new material. I think it's important that these groups be varied as the class progresses, so that each student works with a variety of their peers. Students may pick or be assigned a group. For classes with which I have interacted as the teaching assistant, pair programming was particularly effective for helping students work through complicated concepts together.

Students should be able to gauge their current understanding of the material as a course progresses. I will use in-depth homework assignments to help students interact with and solidify material.

However, I will also use frequent low-stakes assessments, like weekly quizzes, to provide students with quick, formative feedback. These assessments allow students to check their own learning and allow me to asses student progress so that I can adjust accordingly.

To better prepare myself for future classrooms, I also want to evaluate my own performance during and at the conclusion of each class. At the beginning of each course, I will ask students about their expectations of the class and the material. While some subject matter is rigid and must be covered, courses can be tailored to the needs of students. In introductory courses, students may want to understand how the material relates to other topics, both inside and outside Computer Science. Mid-semester, I will ask students to reflect on their experiences and suggest possible alterations. These evaluations, along with homeworks and more frequent assessments, can be used to guide the remainder of the course. I will share the results of these evaluations with students and make adjustments to the course, as necessary. I would like to provide a flexible classroom that can adapt to the needs of a particular group of students, including the ways in which lectures, small group activities, and assignments are structured. At the end of the semester, I will use final student comments and assessments to look critically at the course and plan for future semesters.

As a teacher, I am a learner also—learning about the students and how they best absorb and create new knowledge. One way I demonstrate this to my students is by acknowledging that I sometimes do not have all the answers; in these cases, I am happy to learn new information myself and present it to my students in a later class session. This shows my students that like them, I too am constantly learning. In my experience, students respond more effectively to a teacher with whom they can relate; I would like my students to see me as a resource in the learning process.

By framing the classroom as a learning environment for both the students and the teacher and by encouraging students to get to know one another, I hope to create a climate in which students feel safe and comfortable. Only then can we begin the journey towards new knowledge, together.

Computer Sciences 537: Operating Systems

Spring 2012

`http://pages.cs.wisc.edu/~jacobson/cs537/S2012`

Instructor	**Teaching Assistant**
Emily Jacobson	TBA
✉ `jacobson@cs.wisc.edu`	✉ `tba@cs.wisc.edu`
⌂ 7358 CS	⌂ XX CS
☎ 262-2263	☎ 262-XXXX
OFFICE HOURS	OFFICE HOURS
Monday, Wednesday: 10-11	Tuesday: 2:30-3:30
Thursday: 5-6	Thursday: 1-2
Friday: 3-4	Friday: 10-11
You are welcome to drop by my office at other times; if my door is open, I'll be happy to chat. You may also email me to set up an alternative meeting time.	*Or by appointment.*

Welcome

Welcome to Operating Systems! This is an exciting part of Computer Science and an active area of research. In this course, we'll cover a variety of topics including operating system structure, process and thread scheduling, synchronization, concurrency, memory management, resource management, file systems and storage.

Prerequisites: This course assumes previous material covered in CS 364 and CS 367, including computer organization (processors, memory, I/O devices and data structures). Additionally, all programming assignments will be done in the C programming language. If you are unfamiliar with C, you may find some of the assignments more challenging than intended; please come talk with me if you have concerns about your background knowledge.

Course Meetings

CLASS
Tuesday, Thursday: 1:00pm - 2:15pm
Engineering 3032

DISCUSSION SECTION
Wednesday: 1:20pm - 2:10pm
Psychology 103

This course meets three times a week, for two lectures and one discussion section. Attendance is mandatory, including for the discussion section. The lecture slots will be used for a mixture of lecture and small group activity; discussion sections will be used for weekly quizzes, supplemental material, and to discuss project assignments.

Course Objectives

This course covers a wide range of material. You can expect to learn about the following:

- Abstractions provided by an operating system and the interactions between them, including:
  - Processes, threads, context switching, CPU scheduling
  - Synchronization primitives including semaphores, locks, condition variables, monitors
  - Memory hierarchy, virtual memory and addressing
  - Persistent storage, file system organization, and tradeoffs between different file systems, including NFS, AFS, and LFS
- C programming language
- Project organization, including header files, Makefiles, and version control

Materials

For this course, we'll be using a free textbook developed here at UW:

Operating Systems: Four Easy Pieces
Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau
Available at `http://pages.cs.wisc.edu/~remzi/OSFEP`.

For our programming assignments, we'll be using the C programming language. If you don't already own a C reference book, I'd recommend a classic:

C Programming Language
Kernighan and Ritchie, 2nd Edition, ISBN 0-13-110362-8

Announcements

Important announcements like assignment clarifications will be sent to the class mailing list (`compsci537-1-s12@lists.wisc.edu`) as well as posted to the web page (`http://pages.cs.wisc.edu/~jacobson/cs537/S2012`), so please **read your email frequently**! You are responsible for the material sent to that list. Also note that this list sends to your `.wisc` account, not your CS account.

Assignments

Throughout the semester, you will complete a series of programming assignments to familiarize you with OS concepts. Details will be provided when each assignment is distributed. Programming assignments will be completed in groups of *2 or 3*. Students will choose their own groups, or I can help you find a partner. **Programming assignments make up 40% of your grade**.

Late days: You each have 3 late days to use over the course of the semester. Late days are *24 hour days*. You may use these late days on separate assignments, or combine them for one assignment. After you have used up your late days, *no late assignments will be accepted*. If you have a serious personal emergency that will prevent you from turning in an assignment on time, please contact me; we may be able to make alternative arrangements (at the instructor's discretion).

There will also be several problem sets. These problems are *optional*, but serve as good practice problems for quizzes and the final. It is in your best interest to complete them and ask questions; the quizzes and final exam will likely be challenging if you have not worked through the problems. Both the TA and I will be happy to answer questions and look over homework solutions.

Quizzes and Exams

We will have quizzes and a final exam in this course. Both the quizzes and the final exam will be closed-book and will explore all material covered prior.

Quizzes will occur during the discussion section and are marked on the schedule. **Quizzes make up 30% of your grade**. There will be no makeup quizzes. We will drop your lowest quiz grade.

The final will be on **Sunday, May 13 from 12:25pm to 2:25pm**. **The final exam will make up 30% of your grade**.

Grades

My goal is for everyone in this course to demonstrate mastery of the material we cover. I do not curve grades; I would be pleased if you all do well. Grades will be posted on Learn@UW.

Programming Assignments (40%)	Project 0 Project 1 Projects 2-5 (45 each)	5 15 180
Quizzes (30%)	10 total (15 each)	150
Final Exam (30%)	150	150
Total Points		**500**

Programming assignments will be completed in small groups (2-3). You may chat with other groups about the design process, but you may not share code or algorithms. Quizzes and the final exam will be individual; there should be no communication between students during these assessments.

If you have any questions about what is and is not a permissable activity, please contact me. Consequences for cheating include receiving a 0 on the assignment or in the course; an academic misconduct notation will also appear on your transcript.

Computing Facilities

For this course, we will use the Linux workstations located on the first floor of Computer Sciences. All students registered for this class should have an account; please contact the CSL if you do not.

It is also possible to complete the assignments on your personal computer; you must have the proper compilers and libraries installed.

Cell Phones and Laptops

Cell phones need to remain turned off during class. You may take notes on a laptop during class, but please respect me (and your fellow classmates) by only using your laptop for note-taking activities. If your laptop use is distracting, I will ask you to stop bringing it to class.

Special Learning Needs

We want to fully include persons with disabilities in this course. If you have special circumstances that you believe may affect your performance in this class, please let me know as soon as possible; you should also contact the McBurney Disability Resource Center (`www.mcburney.wisc.edu`). We will work together to ensure you are able to fully participate.

Tentative Course Schedule

This schedule may change. Please consult the course website for the most up-to-date version.

Readings associated with each lecture are listed below in italics. Our textbook is grouped into four conceptual pieces: virtualization (V), concurrency (C), persistence (P), and distribution (D). There is also a short introduction (I) and a series of appendices (A). The section for each reading is indicated along with the title of the reading.

Week	Lecture 1	Lecture 2	Discussion	Turn In
1 1/23/2012	Introductions *I: Intro*	Processes *V: Processes, Process API, Direct Execution*	Intro to C, SVN	**Due 1/27**: P0
2 1/30/2012	Scheduling *V: Scheduling, MLFQ, Lottery*	Threads *C: Intro, Thread API*	Intro to C, Makefiles	**Due 2/3**: P1
3 2/6/2012	Semaphores *C: Semaphores*	Locks *C: Locks, Locks:Usage*	**Quiz 1**: Processes	
4 2/13/2012	Condition Variables *C: Condition Variables*	Monitors *A: Monitors*	**Quiz 2**: Scheduling	
5 2/20/2012	Deadlock *C: Deadlock*	Memory Management *V: Address Spaces, Translation, Segmentation*	**Quiz 3**: Synchronization	**Due 2/24**: P2
6 2/27/2012	Paging *V: Paging*	Topic TBA	**Quiz 4**: Deadlock	
7 3/5/2012	Paging: TLBs *V: Faster*	Page Tables, Cont'd *V: Smaller*	**Quiz 5**: Paging	
8 3/12/2012	Replacement Policies *V: Beyond Physical, Policies*	I/O Disks *P: Devices, Disks*	**Quiz 6**: Page Tables, Replacement	
9 3/19/2012	RAID *P: RAID*	File Systems *P: Files and Directories*	**Quiz 7**: Disks and RAID	**Due 3/23**: P3
10 3/26/2012	FS Implementation *P: FS Implementation*	FFS *P: FFS*	TBA	
Spring Break				
11 4/9/2012	Journaling *P: Journaling*	LFS *P: Log-structured FS*	**Quiz 8**: FFS	
12 4/16/2012	Distributed Systems, NFS *D: Into, Sun's NFS*	AFS *D: Andrew FS*	**Quiz 9**: Journaling	**Due 4/20**: P4
13 4/23/2012	GFS *A: Google FS*	Virtual Machines *A: VMM Intro, VMM-Mem*	**Quiz 10**: AFS, NFS	
14 4/30/2012	Security *D: Security*	Security *D: Authentication, Kerberos*	**Quiz 11**: Virtual Machines	**Due 5/4**: P5
15 5/7/2012	Wrap-Up	Final Review	Final Review	
Final: Sunday, May 13, from 12:25pm - 2:25pm				

Syllabus Grading Rationale

Computer Sciences 537: Operating Systems

As part of my syllabus, I've also created grading criteria and policies. I have chosen to use criterion-referenced grading. As part of my syllabus, I state "I would be pleased if you all [the students] do well." I would like to facilitate a classroom environment in which students strive to master and demonstrate mastery of the material for their own learning benefit, rather than for the sake of competition with fellow classmates. In Chapter 6 of *Teaching at Its Best*, "Testing and Grading," Nilson articulates that criterion-based grading causes final grades to reflect "what students know, compared to the teacher's standards." However, because it can be difficult for new teachers to effectively capture specific grading goals in a new classroom, I also intend to incorporate some aspects of norm-referenced grading into my course. Assessments will be graded based on pre-determined criteria that I define (these will form a rubric that the TA and I will use). Once all grades have been computed, I will evaluate how well students performed. If I determine that the assessment was more difficult that intended, demonstrated by overall low scores, I may choose to "bump" all students' grades up by some number of points. This has the effect of moving the curve of the class, without enforcing a particular kind of curve.

Within my grading scale, I will use a series of cutoffs to translate numeric scores to letter grades. I plan to use the 90%-80%-70%-60% scale. For each assessment, the point breakdown will correlate with this scale, so that a C represents a baseline of material mastery that must be demonstrated, with criterion for B and A adding levels of mastery.

In the syllabus, students are told how the assessments (projects, quizzes, and the final exam) will combine to produce a final grade. As stated earlier, I also clearly tell students that grades will not be curved. I support this level of transparency because I believe it will help create a more open and friendly classroom environment. My assessments are a mixture of individual work (quizzes and the final exam) and cooperative work (projects), and I want students to be driven to excel at both.

For all assessments, there will be many opportunities for partial credit. When points are deducted, students will be given clear feedback with the hopes that they will learn from these mistakes for future assignments. Optional problem sets (not graded) also provide students with copious opportunities for feedback about their work. Problem sets, however, require students to seek out help, since these problems will likely not be covered during class time.

There will not be opportunities for students to revise assignments. However, quizzes are low-stakes assessments with the hopes that students will easily be able to evaluate their own progress and ask for help when necessary. Quizzes comprise 30% of the total grade in the class, so each quiz will be worth approximately 3% of the grade. Thus, students can receive frequent feedback with relatively low stakes. Further, the lowest quiz grade will be dropped from the student's final grade. I believe this has two benefits; it allow students to (1) have an "off" day or misunderstand some material, or (2) simply miss a class. In a large classroom environment and with two concurrent sections of the class, offering make-up quizzes seems difficult and potentially problematic.

To allow for some flexibility in students' schedules, particularly upper-level students who are likely taking other difficult, time-consuming classes concurrently, I provide students with three free late days. They may uses these days on any project, either individually or combined for one project. This allows students to evaluate their schedules and have additional time when necessary. Once they have used all three late days, however, no late assignments with be accepted. I believe this policy gives students the flexibility they may need, while still ensuring that material is covered and assessed on a reasonable schedule.

Overall, I believe there are several ways in which students benefit from my grading policies:

- Criterion-referenced grades encourage students to master material without putting them in direct competition with one another.
- Allowing for scores to be uniformly increased incorporates some positive aspects of norm-referenced grading, such as potentially adjusting grades based on observed difficultly of questions, when this might differ from the intended criterion.
- Partial credit provides students will the opportunity to demonstrate and get credit for what they *do* know.
- Frequent quizzes and dropping the lowest quiz score provide students with low-stakes assessments to continually evaluate their own learning.
- Late days allow students the flexibility to work within their own schedules, while still enforcing due dates.

Unit Plan
Computer Sciences 537: Operating Systems

Unit Plan

The course unit plan is included in the course syllabus, described above.

Explanation of Projects

This course has six projects. Below, each assignment is described, including a brief description of the assignment itself, the material covered, and the goals of the assessment. Students will work in small groups (2-3 students) on all projects except Project 0.

Project 0

This is not a programming assignment. Instead, I ask students to tell me about themselves: what prior Computer Science experience they have, why they are in the class, and if there are any additional things I should know about them as the course moves forward. I will also ask students to provide me with a current picture, so that I can learn students' names. Students are also given an opportunity to ask questions of me.

Project 1

This is a very short programming assignment designed to re-familiarize students with the C programming language. Students will be asked to implement a simple algorithm, likely sorting. The main goal of this assignment is to get students engaged with C and to preview for them the depth of the future assignments. Because C is only taught briefly before this course, students often find later assignments to be more challenging than intended. Students will work in pairs and will be encouraged to practice good pair programming techniques. Project 1 will be due before the university drop date, so that students who decide the material is too difficult have a chance to change their course schedule before an official withdrawal is noted on their transcript.

Project 2

This is the first real Operating Systems programming assignment. Students will build a shell program. The main goal of this assignment is to understand processing forking. Further, students will learn about error handling and defensive programming.

Project 3

In this project, students will explore a variety of synchronization primitives and use the POSIX Threads (pthreads) library. The goal of this assignment is to have students to gain experience with pthreads, mutexes, and condition variables.

Project 4

In this project, students will evaluate the performance of several scheduling algorithms. Students will be given some basic code with a simple simulator, and will implement several scheduling algorithms within this framework. Then, students will perform basic benchmarking to evaluate the relative performance. The goal of this assignment is to solidify these scheduling algorithms for student and to expose students to basic benchmarking practices.

Project 5

In this project, students will build a very simple distributed file system. The main goal of this assignment is to have students explore issues related to distributed protocols, including idempotent operations.

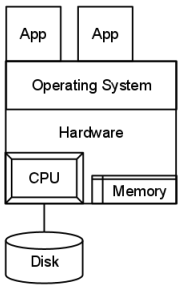
Lesson Plan
Computer Sciences 537: Operating Systems
Lecture 2: Processes

General Lesson Structure

This class session will consist primarily of chalkboard-style lecture, with a small group activity in the middle. Students will be broken into small groups (2-3 students) for a 7-10 minute exercise (on the handout).

Introduction

This is the first "real" day of our Operating Systems course. Today, we're going to talk about the process, a main abstraction provided by the operating system. Before we dive in, let's remember where we are in the world.

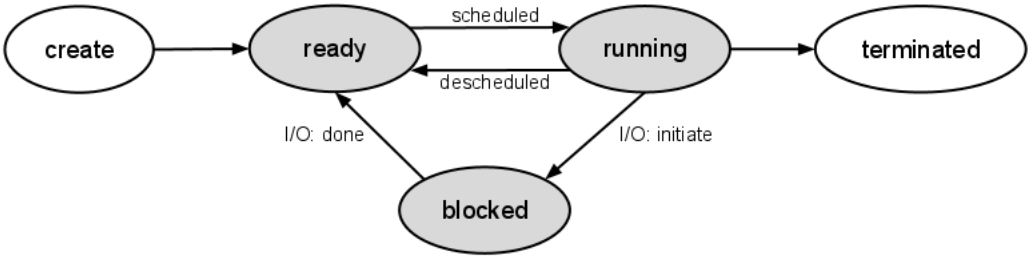


You'll remember this diagram from your computer organization course—354 if you took that here at Wisconsin. In this course, we'll focus on the operating system itself. On top of the operating system, we run one or more applications. Now, each application is a "process" and is a running instance of some program. You've written plenty of programs. Now, when you run a program, you're actually running a process. In fact, you could run multiple instances, processes, of a single program. For instance, if you're watching a video, and you want to open another video at the same time, you're likely opening a second copy of the program. So there isn't a 1:1 relationship between a program and the process executing. What we'll focus on today is how these processes get created, and how the operating system manages the processes and decides which process to run.

Main Concepts

- review of computer organization
- transformation from written code (in this course, in C) to a program on disk
- process as a fundamental OS abstraction; running instance of a program
- what a process looks like loaded into memory (stack, heap, .bss, code)
- CPU virtualization
  - policy: scheduling (referenced here, covered in a later lecture)
  - mechanism: context switches (referenced here, covered in a later lecture)

- process execution states



- OS data structures
  - ready queue
  - blocked queue
  - running process
- process API
  - `fork()`
  - `exec()`
  - `wait()`

Goals for Students

- differentiate between a process and a program
- complete execution state diagram, explain states
- complete example scheduling for sample set of processes
- effectively use process API calls to create new processes and wait for their termination

Short-term Assessment Notes

One form of assessment for this lecture will be in the form of a weekly quiz (included in this portfolio). This quiz will ask students to think about what the visible results will be for simultaneous processes.

Further Assessment Notes

The first real OS programming assignment (Project 2) will have students further explore these concepts. This project, in which students will implement a simple shell, will rely on the process API presented in this lecture and give students hands-on experience creating new processes and managing child processes.

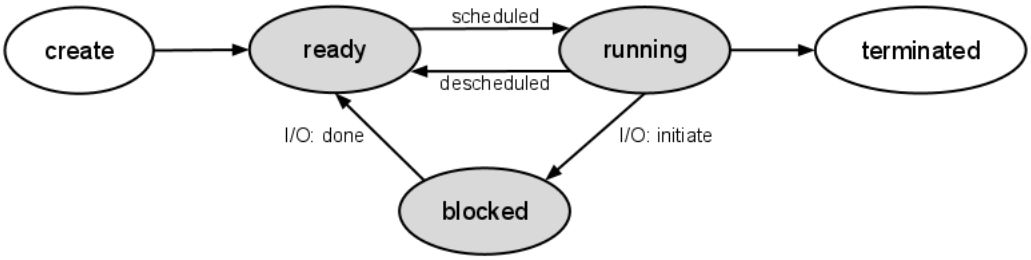
Additional Materials

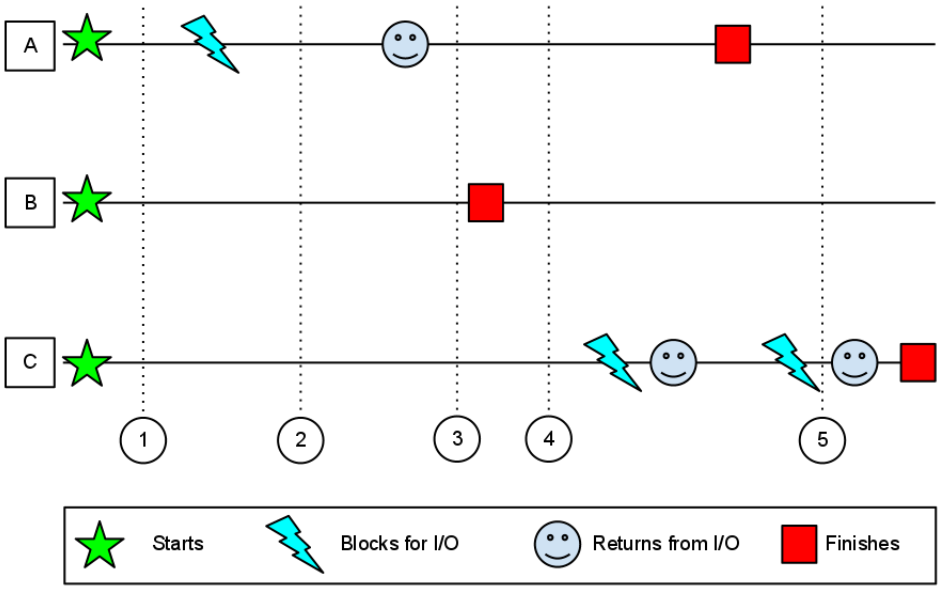
- My lecture notes: these would be posted to the course website after the lecture
- Handout: this will be used for the interactive activity and to aid in student note-taking

Lecture Notes
Computer Sciences 537: Operating Systems
Lecture 2: Processes

- quick review of basic computer organization, including CPU, memory, and disk
- you create a program with some set of goals
- a program is static: it's just a set of instructions on disk
- the running instance of a program is called a process
- the process is one of the most basic abstractions provided by the operating system
  - e.g., you're watching a movie, and you want to open a second one; one program, two processes
  - code (machine instructions), data (initialized, dynamically allocated, stack variables)
- In the misty eons of time, computers were uniprocessors: only a single process was run at once. When that process completed, another process was launched.
- However, the uniprocessor gave way to the multiprocessor era, in which we'd like to run many processes "simultaneously": we only have one CPU though
- We virtualize the CPU, so each process thinks it has full control of the CPU
- Thus, we run multiple processes "at once," but really they are time sharing *one* physical CPU
- What actually happens is that we'll run one process for some amount of time, then "pause" and run another process for some period of time
  - *We now have multicore machines now, on which we could actually run multiple processes* ***at once****. However, we'll focus on single-CPU machines for the moment (and for most of this course).*
    - two major components here, both of which we'll come to later
    - how this scheduling (policy) happens is something well get to in another lecture
    - context switches (mechanism)
      - how do you switch between two active processes?
      - what state must be saved?
    - fundamental concept: separate mechanism (how) and policy (when/why)
- process execution state (simplistic view: once a process has been created)
- once a process is running, it may want to interact with another device: to read from the disk, for instance. We call these interactions Input/Output = I/O

At this point, students will participate in a small-group activity (see handout). Students will work for a few minutes (7-10), after which we will come back together as a class to go through the activity as a class.





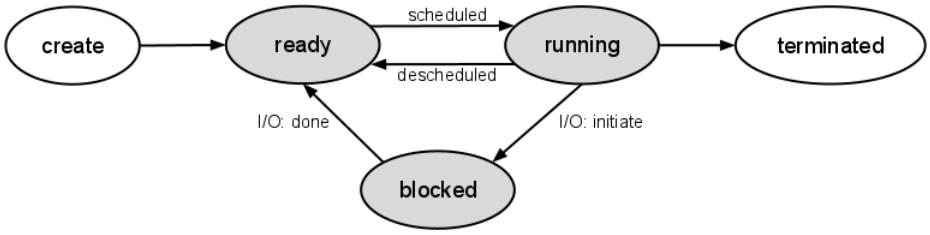
Time	Ready Queue	Running	Blocked Queue	Completed
1	B, C	A		
2	C	B	A	
3	A, C	B		
4	A	C		B
4 (alternate)	C	A		B
5			C	A, B

The alternate time=4 is made possible by different scheduling algorithms. If we choose to run C after B completes, this is round robin scheduling, in which we cycle through the processes, in order, looking for the next ready process. If we choose to run A instead, this is priority scheduling, where we pick the highest priority process to run next (here, priority = alphabetically first). Both of these are scheduling without preemption, meaning that a process runs until it finishes or blocks.

Handout

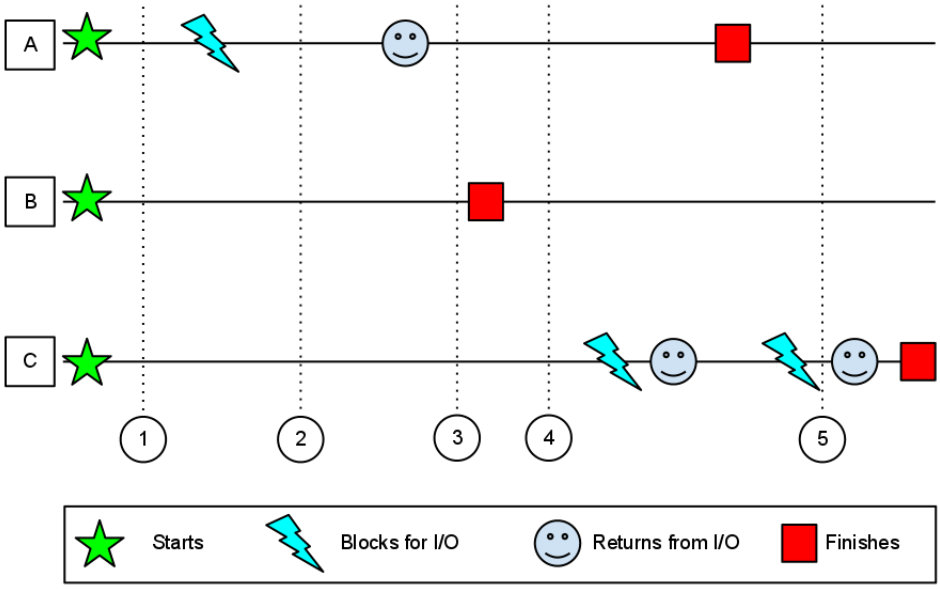
Computer Sciences 537: Operating Systems

Lecture 2: Processes



Exercise

There are three processes executing on a single CPU. Given an execution ordering, fill in the relevant data structures the operating system uses to track process information for each marked time. Note that these time values represent snapshots of the system; there is no corresponding change in execution state at any of the marked times.



Time	Ready Queue	Running	Blocked Queue	Completed
1				
2				
3				
4				
5				

Assessment

Computer Sciences 537: Operating Systems

Lecture 2: Processes

The goal of this assessment is for students to demonstrate their (route) understanding of process execution states (LO thinking) and to think about potential schedules that can result from multiple processes being run concurrently (HO thinking). This would be a quiz during discussion section; when given to students, room would be provided beneath each question for their answer.

Part 1

What are the possible execution states for a process (you need not include `create` or `terminated`) Hint: there are 3 states. For each state, please briefly explain (1) what being in this state means for a process, and (2) how a process *enters* and *exits* this state.

Part 2

Describe all possible outputs from this program.

```
int a = 0;

pid_t pid = fork();
if (pid == 0) {
    a = 1;
    printf("%d", a);
} else {
    printf("%d", a);
}
```

Part 3

Now there are two concurrent processes. You don't know what order they will be scheduled in, or when the dispatcher will deschedule one process and schedule the other. Describe all possible outcomes from this program.

Initialization

```
int x = 0;
int y = 0;
```

Process 1

```
for ( ; x < 4; x++) {
    y = 0;
    printf("%d", x);
    y = 1;
}
```

Process 2

```
while (x < 4) {
    if (y == 1) {
        printf("a");
    }
}
```

Rubric for Assessment

Computer Sciences 537: Operating Systems

Lecture 2: Processes

This rubic will be used to grade the quiz. Although each part is broken down into several point categories, students will also be granted partial credit within a category when appropriate.

Part 1	
Identify each state (.33 each)	1
State meaning (.5 each)	1.5
Enter/exit explanation (.5 each)	1.5
Part 2	
2 outputs listed	1
01 output	2
10 ouptut	2
Part 3	
Correct pattern, without multiple a's (0-1-2-3, with or without a's)	3
Recognizing 0 or more a's between 0-1-2-3	3
Total	**15**

Microteaching Experience

Reflection

For my microteaching experience, I selected material usually covered in an upper-level undergraduate Operating Systems course (537 here at Wisconsin). In particular, I chose material usually covered on the second day of class: the abstraction of a process running on a CPU.

During my first teaching experience, there were a few things that went well. I was able to use a bit of humor—referencing the "misty eons of time"—to get students intrigued about the topic. Students commented that they found the lecture interesting, and they were able to answer questions I posed during a group activity.

However, there were several aspects of this first experience that were less ideal. My lecture pace was too quick, and I was not careful about using appropriate jargon; in the future, I need to carefully pace myself and make sure to use language with which students will be familiar, rather than language I use as a researcher. I (and the students) were unhappy with the interactive activity. I went through the first few iterations and then asked students to volunteer answers. However, because the activity was simplified, some students were reluctant to volunteer information because the answer seemed "too obvious." I also used language that prompted responses, which several students found frustrating. Although I knew the activity might be problematic from the outset, I appreciated the opportunity to try something new, and I found the comments from my peers to be very constructive.

The first round of microteaching convinced me that I need to think more carefully about how to structure a lecture and activity to engage students effectively while covering an appropriate amount of material. In round two of microteaching, I had several new goals. I wanted to be more careful about my pace, making sure to motivate the topic and focus on key details. I also modified the activity, so students would work on a handout in pairs and then discuss the answers in a large group. The handout also provided students with an important diagram that was presented in lecture; I wanted to give students the diagram so they didn't have to quickly copy it down and could instead think about what other notes they might want to take in addition.

I thought my teaching was much more effective during the second round. Students appreciated the handout and the small-group activity. The first half was heavy on lecture; in the future, I'd like to update this to be more discussion based. In a real class, students would have had a reading

assignment prior to class, so lecture would be an interactive review before we moved on to more advanced topics. I was very happy with the short activity. Although some students were confused, they had time to think about the exercise in small groups and clarify confusions with each other before we went through the exercise as a whole class. Trying this out in a safe setting helped me envision how such collaborative learning could be incorporated in a future classroom.

As a whole, I thought the microteaching experience was exceptionally valuable. I appreciated the opportunity to teach in front of willing subjects, and I felt comfortable trying techniques (including the group activity) that I haven't seen done in real versions of this course. I found great value in the feedback I got from my peers, both about the delivery itself, and also about technical details like that students might appreciate handouts or posted lecture notes for diagram-heavy topics. Further, I thought microteaching was valuable because of the twelve other lessons I had the chance to observe and critique. As a group, we were able to think critically about the effectiveness of various teaching methods, both within our own disciplines and beyond.

Teaching Portfolio Acknowledgements

The course described in this portfolio was influenced by prior offerings of similar courses:

- Jerod Weinman's CSC 213 at Grinnell College, Fall 2008 (I was a student in this course)
  `http://www.cs.grinnell.edu/~weinman/courses/CSC213/2008F/`
- Janet Davis's CSC 213 at Grinnell College, Fall 2010
  `http://www.cs.grinnell.edu/~davisjan/csc/213/2010F/`
- Bart Miller's CS 537 at University of Wisconsin-Madison, Spring 2011
  `http://pages.cs.wisc.edu/~bart/cs537.html`
- Remzi Arpaci-Dusseau's CS 537 at University of Wisconsin-Madison, Fall 2011
  `http://pages.cs.wisc.edu/~remzi/Classes/537/Fall2011/`