Assignment #1

Benchmarking Operating Systems

- See the description at http://www.cs.wisc.edu/~cao/cs736/assignment1.html

Message Passing

- send: ! recv: ? e.g. {west ? c -> east ! c}
- guarded commands
- alternative commands
- command lists
- repetitive commands

Example

- messages:

```
*[
X?has(n) -> search;
   X!(I<size);
X?insert(n) -> search;
   if (not present) add;
]
```

- procedure calls:

```
entry procedure has (n) :
   search; return(I<size);
entry procedure insert(n)
   search; if (not present)
   add;
```

Using CSP for Synchronization

- semaphores
- bounded buffer

Examples of CSP

- disk scheduler

Two Paradigms

- shared memory
- message passing

Equivalence of the two paradigms

- synchronization
- communication

Duality of the Paradigms

- real life examples?
- message-oriented system
- procedure-oriented system

Synchronization in UNIX kernels

- uniprocessor kernel
- SMP kernel
- client/server architecture
- message-passing machines

Examples of CSP

- The sieve of eratosthenes
  - sieve(I-1)?p; print!p;
    mp :=p;
    *[sieve(I-1)?m -> *{m>mp -> mp:=mp+p];
    [m = mp -> skip;
    [] m<mp -> sieve(I+1)!m;]
    ]

The Process Abstraction

- process state
  - user running, kernel running, ready, asleep, *zombie*
- process context
  - user address space, control info, credentials, env variables, and hardware context
- user credentials
  - real/effective uid/gid; setuid;
- Access rights based on user
  - a simplification that causes many problems

Process Control Table

- struct task_struct (*include/linux/sched.h*)
  - identification: p_pidp, p_pgidp
  - pointer to user structure: struct user p_user
  - current state: p_stat, p_wcode, p_wdata
  - link pointers for scheduler queue or sleep queue
  - hierarchy info: p_parent, p_child, p_sibling
  - mmu info: struct as *p_as
  - real and effective ids: struct cred *p_cred;
  - scheduling priority and related info;
  - singal handling information: masks of signals ignored, blocked, etc
  - pointers to link active, free, or zombie processes; misc flags

u area

- struct user (*/usr/include/sys/user.h*)
  - process control block (hardware context)
  - pointer to proc
  - user credentials
  - arguments, return value, or error from syscalls
  - program info (*u_tsize*, *u_dsize*)
  - open file descriptor table (*u_nofiles*, *u_flist*)
  - pointers to vnode of directories (*u_cdir*, *u_rdir*, *u_ttyvp*)
  - resource statistics (*u_rlimit*, *u_ticks*)
  - kernel stack

Fork

- do_fork (kernel/fork.c)
- lock_kernel (include/asm_i386/smp_lock.h)

Process Creation: *fork*

1. Reserve swap space for the child's data and stack;
2. allocate a new PID and proc structure for the child
3. initialize the child's proc structure
4. allocate address translation maps for the child
5. Allocate the child's u area and copy the content from the parent
6. update the u area to refer to the new address space and swap space
7. Add child to the set of processses sharing the text region of the program that the parent is executing

*8. duplicate the parent's data and stack region one page at a time and update the child's address maps to refer to these new pages

9. acquire references to shared resources inherited by the child, such as open files and the current working directory

Fork

10. initialize the child's hardware context by copying a snapshot of the parent's registers
11. make the child runnable and put it on a schedule queue
12. arrange for the child to return from fork with a value of zero
13. retuen the pid of the child to the parent

Solaris 2.4:
steps 2, 3, 5 and 10: *common/os/fork.c: getproc*
step 4: *as_alloc*
steps 7, 8: *common/vm/vm_as.c:as_dup, seg_dup*
steps 9: *bump_fcnts, shmfork*
step 11, 12, 13: *common/os/fork.c: fork*

Threads and LWPs

- Motivation
- Threads
  - a process = a set of threads + resources
  - shared resource: an address space, open files, user credentials, quotas, etc.
  - private objects: PC, stack, register context
- user threads and kernel threads
- lightweight processes

Kernel Threads

- common/os/disp/thread.c, common/sys/thread.h
- resouces used by kernel threads:
  - saved copy of the kernel registers
  - priority and scheduling information
  - pointers linking scheduler queue or resource wait queue
  - pointer to stack
  - pointer to assocaited lwp and proc structure
  - pointer to maintain a queue of all threads of a process and all threads in the system
  - information about associate lwp

LWP Implementation

- common/os/lwp.c
- each lwp is associated with a kernel thread:
  - system call arguements, results and error code
  - signal handling info
  - resource usage and profiling data
  - virtual time alarms
  - user time and CPU usage
  - pointer to kernel thread
  - pointer to proc structure
- design issues: fork, system calls, signals

User Thread Library

- Solaris thread library (*/usr/include/thread.h*)
  - thr_create, thr_join, thr_exit;
  - thr_setprio, thr_getprio, thr_yield;
  - mutex_init, mutex_lock, mutex_unlock;
- C++ task library