CS 537: Introduction to Operating Systems (Summer 2017)

University of Wisconsin-Madison
Department of Computer Sciences

Midterm Exam 2

July 21st, 2017

3 pm - 5 pm

There are **sixteen (16) total numbered pages** with **ten (10) questions**.

PLEASE READ ALL QUESTIONS CAREFULLY!

There are many easy questions and a few hard questions in this exam. You may want to use a **easiest-question-first scheduling policy**. This will help you to answer most questions on this exam without getting stuck on a single hard question. The last 2 questions are worth 20 points each. All other questions are worth 10 points each.

Good luck with your exam!

Please write your **FULL NAME** and **UW ID** below.

NAME:_____

UW ID:_____

Grading Page

Question	**Points Scored**	**Maximum Points**
1		10
2		10
3		10
4		10
5		10
6		10
7		10
8		10
9		20
10		20
Total		**120**

1. Page Replacement Policies

a. Consider the following request sequence of virtual pages

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

For each replacement policy below, give the number of hits and the virtual page numbers remaining in physical memory at the end of this request sequence.

The number of physical frames = 4

Policy	**# of Hits**	**VPNs remaining at the end**
OPT		
FIFO		
LRU		

b. Write a **sequence of 10 virtual page requests** that has a **hit rate of zero** with **LRU** page replacement policy.

Virtual Pages available: 0, 1, 2, 3, 4
The number of physical frames = 4

c. What are the **disadvantages** of using the following page replacement policies in **real-world** systems?

i. **OPT:**

ii. **FIFO:**

iii. **LRU:**

2. Hardware Locks

Suppose we have a new instruction called **`CompareAndRestore (CAR)`**, and it does the following **atomically** (here is the C pseudo-code):

```
int CompareAndRestore(int *ptr, int expected, int new) {
    int original = *ptr;
    if (original != expected)
        *ptr = new;
    return original;
}
```

a. Implement a working **spin-lock** using the CompareAndRestore (CAR) instruction.

```
typedef struct __lock_t {
    int isFree;
} lock_t;

void init(lock_t *lock) {
    lock->isFree = 1;
}

void acquire(lock_t *lock) {




}

void release(lock_t *lock) {



}
```

b. How would you **evaluate** your lock based on the following 2 criteria?

i. **Fairness:**

ii. **Performance:**

3. Segmentation + Paging

In this question, we consider address translation in a system which uses a **hybrid of segmentation and paging** for memory management. There are **three segments** namely, code, heap, and stack. The **two higher-order bits** (MSBs) in the virtual address are used to identify the segment. 00 for code, 01 for heap, and 11 for stack.

Parameters and Assumptions:

- Size of virtual address space = 1 KB
- Page size = 16 bytes
- Size of physical memory = 4 KB
- Size of one Page Table Entry (PTE) = 2 bytes
- The virtual pages with VPNs 0, 1, 16, 17, 18, and 63 are the ONLY valid pages.

Answer the following questions based on the parameters and assumptions described above.

1. Number of **bits** needed for the Virtual Page Number (**VPN**): ————————
2. Number of **valid virtual pages** in the **code** segment: ————————
3. Value of the **bounds register** for the **heap** segment: ————————
4. Number of **PTEs** in the **stack** segment's **page table**: ————————
5. **Total size** of **ALL** the **page tables** used in this system: ————————

4. Locked Data Structures

Assume you have the following code for removing the head of a **shared linked list**. Assume each line is performed **atomically**. Assume a list `L` originally contains nodes with keys 1, 2, 3 and 4. Now there are **two threads** `T` and `S` that are popping the list concurrently.

```
typedef struct __node_t {
    int key;
    struct __node_t *next;
} node_t;

typedef struct __list_t {
    node_t *head;
} list_t;

int pop(list_t *L) {
    if (!L->head) return -1;            // line 1
    int rkey = L->head->key;            // line 2
    L->head = L->head->next;            // line 3
    return rkey;                        // line 4
}
```

a. Given the following sequences, fill in the results. The sequence contains `T` and `S`, designating that one line of C-code was scheduled for the corresponding thread. For example, a sequence of `TTTSS` indicates that 3 lines were run from thread `T` followed by 2 lines from thread `S`. You should assume that each sequence is executed independently. In other words, the state of the linked list is the same (with 4 nodes) at the start of each sequence. The right most column in the table below represents the value of `L->head->key` at the **end** of the sequence.

Sequence	`rkey` from `T`	`rkey` from `S`	`L->head->key`
TTSSSSTT			
TTTSSSST			
TTSSTTSS			
TSSSTTTS			

b. In the pop() method given above, which **line(s)** of code form the **critical section**? Our goal here is to **maximize the concurrency** among threads that are trying to pop from this shared linked list.

5. TLB, Memory, and Page Faults!

In this question you will examine virtual memory reference traces. An access can be a TLB Hit or a TLB Miss; if it is a TLB miss, the reference can be a page hit (page present in physical memory) or a page fault (page not present in physical memory).

Assume a **TLB with 2 entries** and a **memory that can hold 4 pages**. Assume the TLB and memory are initially **empty**. Finally, assume **LRU replacement** is used for both TLB and memory.

Below each virtual memory reference, **mark** if the reference is a:

- TLB Hit (**H**), or
- TLB Miss followed by a page hit (**M**), or
- TLB Miss followed by a page fault (**F**)

Also, write the **contents** of the TLB and the Memory at the **end** of each virtual memory trace.

Virtual Memory Reference	**TLB**	**Memory**
0, 1, 2, 3, 4, 5, 6, 7		
0, 1, 2, 3, 0, 1, 2, 3		
0, 1, 2, 3, 4, 0, 1, 2		
3, 7, 3, 7, 1, 3, 1, 7		

6. Inverted Page Tables

Assume a system that uses an **Inverted Page Table (IPT)** for memory management. Remember, that ALL processes in the system will share the same page table in this case.

Parameters:

- Size of virtual address space = 32 KB
- Page size = 4 KB
- Size of physical memory = 64 KB

Given below (on the left) are the contents of the **physical memory** (starting from physical frame 0 down to the max size). The entry in the physical frame 0 (i.e., P3 (VPN:1)) means that this physical frame contains the virtual page of process P3 with VPN = 1.

Draw a **diagram** of the **Inverted Page Table (IPT)** and write the contents of the Page Table that reflects the state of the physical memory shown below. Label the **fields** of the IPT properly. It is enough to show only the **essential fields** in your IPT that are needed for this scheme to work.

Physical Memory Contents **Inverted Page Table**

P3 (VPN:1)
P1 (VPN:2)
FREE
P3 (VPN:0)
P1 (VPN:5)
FREE
P0 (VPN:2)
P2 (VPN:6)
P1 (VPN:3)
FREE
P3 (VPN:7)
FREE
P1 (VPN:4)
FREE
FREE
P0 (VPN:1)

7. Condition Variables

Assume the following implementation for the famous **producer/consumer** problem. You may assume that the code compiles and executes successfully.

```
void put(int value) {
  buffer[fillptr] = value;
  fillptr = (fillptr + 1) % max;
  count++;
}
```

```
int get() {
  int tmp = buffer[useptr];
  useptr = (useptr + 1) % max;
  count--;
  return tmp;
}
```

```
void *producer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    P_mutex_lock(&mutex);          //p1
    if (count == max)              //p2
      P_cond_wait(&empty,&mutex);//p3
    put(i);                        //p4
    P_cond_signal(&fill);          //p5
    P_mutex_unlock(&mutex);        //p6
  }
}
```

```
void *consumer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
    P_mutex_lock(&mutex);          //c1
    if (count == 0)                //c2
      P_cond_wait(&fill,&mutex);//c3
    int tmp = get();               //c4
    P_cond_signal(&empty);         //c5
    P_mutex_unlock(&mutex);        //c6
    printf("%d\n", tmp);
  }
}
```

Assume further that the only way a thread stops running is when it explicitly blocks in either a condition variable or lock (in other words, no untimely interrupts switch from one thread to the other).Also assume there are NO SPURIOUS WAKEUPs from wait().

a. In the following, show which lines of code (from p1 - p6 and c1 - c6) run given a particular scenario. Scenario 0 is completed for you as an example.
   Scenario 0: 1 producer (P), 1 consumer (C), max = 1. Producer P runs first. Stop when consumer C has consumed one entry.
   P: p1, p2, p4, p5, p6, p1, p2, p3
   C: c1, c2, c4
   Scenario 1: 1 producer (P), 1 consumer (C), max = 1. Consumer C runs first. Stop when consumer C has consumed one entry.
   P:
   C:
   Scenario 2: 1 producer (P), 2 consumers (Ca, Cb), max = 1. Consumer Ca runs first, then P, then Cb. Stop when each consumer has consumed one entry.
   P:
   Ca:
   Cb:
b. Are there any **bugs** in this implementation? If so, how do you **fix** them?

8. Advanced Locks!

Consider the following implementation for a lock in Solaris using **Queues, Test-And-Set, Yield, and Wakeup**.

```
typedef struct __lock_t {
    int         flag;
    int         guard;
    queue_t     *q;
} lock_t;

void lock_init(lock_t *lock) {
    lock->flag = lock->guard = 0;
    lock->q    = queue_init();
}

void lock(lock_t *lock) {
    while (xchg(&lock->guard, 1) == 1)
        ; // spin
    if (lock->flag == 0) {
        lock->flag = 1;
        lock->guard = 0;
    } else {
        queue_push(lock->q, gettid());
        setpark();
        lock->guard = 0;
        park();
    }
}

void unlock(lock_t *lock) {
    while (xchg(&lock->guard, 1) == 1)
        ; // spin
    if (queue_empty(lock->q))
        lock->flag = 0;
    else
        unpark(queue_pop(lock->q));
    lock->guard = 0;
}
```

The following are the definitions of the key routines:

- **park()**: Puts the calling thread to sleep.
- **unpark(threadID)**: Wake up the thread with the given threadID.
- **setpark()**: A thread indicates that it's about to park.
- **xchg(int *addr, int new)**: Atomically sets the `new` value in the memory location pointed to by `addr`, and returns the old value at that memory location.

a. What is the **purpose** of the **guard lock**? What may happen if we don't have the **guard lock**? **Explain** with a simple **example**.

b. Assume thread 1 is currently holding the **flag lock** and is inside the **critical section**. During this time, there are **9 more threads** that are currently waiting in the queue for the lock to be released. You may assume that there are no more threads waiting for this lock. You may also assume that all these threads will acquire the flag lock only once. Under this scenario, how many times will the flag lock be **released** by these 10 threads? **Explain** the reasoning behind your **answer**.

c. What will happen if **park()** is called **before releasing the guard lock**?

d. This lock still **spins** while trying to acquire the **guard** lock. So, is this better in any way than simple spin locks with respect to **performance**? Explain your answer.

9. Threads vs Processes!

Assume that the code snippet below compiles successfully, all the APIs like **pthread_create**() do not fail, and the values in the **malloc**'ed memory are all **initialized to 0**.

```
void worker(int *balance) {
    int *counter = malloc(sizeof(int));
    for (int i = 0; i < 1000000; i++) {
        ( *balance )++;
        ( *counter )++;
    }
    printf("balance : val %d, addr %p\n", *balance, balance);
    printf("counter : val %d, addr %p\n", *counter, counter);
}

int main() {
    int *balance;
    balance = malloc(sizeof(int));
    pthread_t t[2];
    for (int i = 0; i < 2; i++) // Creating new threads
        pthread_create(&t[i], NULL, worker, balance);
    for (int i = 0; i < 2; i++)
        pthread_join(t[i], NULL);
}
```

a. What are the **values** of the 2 variables (`balance` and `counter`) after the 2 **threads** (t1 and t2) finish execution? Value here means the contents printed using the following print statements. If the value of a variable may be different in different runs of the program, you should write N/A.

```
printf("%d\n", *balance);   printf("%d\n", *counter);
```

Value	t1	t2
balance		
counter		

b. Consider the **Virtual Addresses (VA)** printed using the following print statements in the 2 **threads** (t1 and t2). **PA** stands for **Physical Address**.

```
printf("%p\n", balance);   printf("%p\n", counter);
```

i. VA of balance in t1 == VA of balance in t2? (TRUE / FALSE)

ii. VA of counter in t1 == VA of counter in t2? (TRUE / FALSE)

iii. PA of balance in t1 == PA of balance in t2? (TRUE / FALSE)

iv. PA of counter in t1 == PA of counter in t2? (TRUE / FALSE)

Now assume that the code given below compiles successfully, all the APIs like **fork**() do not fail, and the values in the **malloc**'ed memory are all **initialized to 0**.

```
void worker(int *balance) {
    int *counter = malloc(sizeof(int));
    for (int i = 0; i < 1000000; i++) {
        ( *balance )++;
        ( *counter )++;
    }
    printf("balance : val %d, addr %p\n", *balance, balance);
    printf("counter : val %d, addr %p\n", *counter, counter);
}

int main() {
    int *balance;
    balance = malloc(sizeof(int));
    for (int i = 0; i < 2; i++) { // Creating new processes
        if (fork() == 0) {
            worker(balance);
            exit(0);
        }
    }
    for (int i = 0; i < 2; i++)
        wait(NULL);
}
```

c. What are the **values** of the 2 variables (`balance` and `counter`) after the 2 **processes** (p1 and p2) created using **fork**() finish execution? Value here means the contents printed using the following print statements. If the value of a variable may be different in different runs of the program, you should write N/A.

```
printf("%d\n", *balance);   printf("%d\n", *counter);
```

Value	p1	p2
balance		
counter		

d. Consider the **Virtual Addresses (VA)** printed using the following print statements in the 2 **processes** (p1 and p2). **PA** stands for **Physical Address**.

```
printf("%p\n", balance);   printf("%p\n", counter);
```

i. VA of balance in p1 == VA of balance in p2? (TRUE / FALSE)

ii. VA of counter in p1 == VA of counter in p2? (TRUE / FALSE)

iii. PA of balance in p1 == PA of balance in p2? (TRUE / FALSE)

iv. PA of counter in p1 == PA of counter in p2? (TRUE / FALSE)

10. Multi-level Page Tables!

Assume a system with a 2-level page table.

Parameters:

- page size = 32 bytes
- virtual address space size = 32 KB
- physical memory size = 4 KB
- Size of one Page Directory Entry (PDE) = 1 byte
- Size of one Page Table Entry (PTE) = 1 byte
- Value of Page Directory Base Register (PDBR) = 30 (decimal) [This means the page directory is held in this page]

The **format** of the **PDE** and the **PTE** is simple. The **high-order (left-most) bit** is the **VALID** bit. If the bit is 1, the rest of the entry is the PFN. If the bit is 0, the page is not valid.

You are given two pieces of information to begin with. First, you are given the value of the page directory base register (**PDBR**), which tells you which page the page directory is located upon. Second, you are given a **complete dump of each page of physical memory** in the next 2 pages. A page dump looks like this:

```
page   0: 0d 0f 06 12 1d 0c 10 03 08 14 03 ...
page   1: 0e 0d 1b 19 0a 0c 12 1b 06 0c 02 ...
page   2: 00 00 00 00 00 00 00 00 00 00 00 ...
...
```

which shows the 32 bytes found on pages 0, 1, 2, and so forth. The first byte (0th byte) on page 0 has the value 0x0d, the second is 0x0f, the third 0x06, and so forth.

For each **virtual address**:

- write down the **physical address** it translates to AND the **data value** at this physical address, OR
- if it is **a segmentation fault** (an out-of-bounds address) write the **reason** for this segmentation fault (**Invalid PDE** OR **Invalid PTE**).

Write all answers in **hexadecimal**.

Virtual Address	**Physical Address OR Seg Fault**	**Data Value**	**Reason for Seg fault**
0x1ebe			
0x45b0			
0x7bb9			

Physical Memory Dump

```
page   0: 0d 0f 06 12 1d 0c 10 03 08 14 03 1b 1c 03 1d 0b 17 17 09 14 14 18 08 17 1d 14 10 03 0f 0a 16 15
page   1: 0e 0d 1b 19 0a 0c 12 1b 06 0c 02 13 00 1c 10 11 02 07 0e 1a 10 08 1e 14 10 06 09 1b 04 10 13 0b
page   2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   3: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   5: 18 09 09 13 12 0f 1a 10 0a 01 11 0b 10 0a 00 01 04 02 1a 12 07 16 13 01 17 07 1e 04 08 07 1a 19
page   6: 02 09 07 18 0c 08 01 0d 13 14 1c 19 07 04 17 18 0f 19 11 08 05 00 13 1b 1c 0e 14 1c 19 12 0c 10
page   7: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   8: 05 13 00 03 18 1b 1a 19 11 1b 15 11 0d 00 02 07 0e 03 1e 11 16 1b 07 1b 1d 08 03 18 1b 18 0c 12
page   9: 07 06 04 1e 1c 1c 11 03 0d 01 1c 08 08 1d 06 1c 09 14 03 1b 0c 0a 14 12 07 03 11 00 1b 0a 05 0a
page  10: 19 00 11 0f 01 0d 1b 13 09 19 0b 1b 0c 02 1e 02 07 17 03 10 1a 12 0a 17 19 11 13 09 0c 1e 1e 00
page  11: 7f 7f 7f 7f 7f 7f ed 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  12: 06 1d 0d 13 09 0f 0b 0a 12 12 1c 15 1c 1a 1e 14 05 1a 1d 1c 11 16 06 19 11 08 01 1a 1e 02 18 0c
page  13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  17: 7f 7f b0 8c 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 96
page  18: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  19: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f2 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  20: 04 13 1b 04 00 02 18 0c 0e 1e 12 08 15 00 08 15 05 01 10 0a 1e 03 18 0c 18 0d 14 0a 06 0e 18 0a
page  21: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c0 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  22: 1e 14 14 04 01 17 05 05 19 1b 01 1d 10 0b 03 0c 1e 08 04 04 03 0a 1c 02 05 1d 1e 1e 07 0a 1d 0f
page  23: 14 13 05 1b 14 03 1c 04 0a 16 12 11 14 0e 09 05 02 16 17 10 16 0b 08 0b 1b 01 0f 07 17 0a 0a 0c
page  24: 06 05 0c 05 00 03 05 0f 01 00 0d 0c 15 01 0c 0c 00 1d 06 02 0e 05 13 0a 03 01 16 08 18 0e 1c 09
page  25: 02 17 11 09 1b 1b 10 1e 19 17 09 00 15 16 0a 0e 11 07 15 11 0b 03 11 09 16 05 0c 06 0c 11 03 01
page  26: 7f ae 7f 7f 7f 8a 7f 7f 7f 7f 7f 7f c1 7f 7f 7f 7f 7f 7f f4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  27: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  28: 18 06 05 17 05 1e 0e 1b 19 06 18 0c 06 01 15 11 09 19 06 0d 02 0a 1b 0f 04 07 13 08 1b 01 1a 13
page  29: 00 14 0f 06 11 0e 1d 1b 12 1b 1b 05 19 19 1e 17 0c 06 14 06 17 11 19 0b 09 1c 14 0b 11 0d 13 10
page  30: ba 7f fa be e2 c3 e0 e8 b5 bd c5 f1 7f a6 e9 95 7f cd a7 d3 93 aa fd 7f 9a 7f d2 f7 91 ab 7f 8b
page  31: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  33: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  34: 13 0a 02 15 1e 12 00 00 02 05 01 01 14 1b 06 1e 0e 18 1c 1d 01 09 12 1d 02 09 1a 03 0c 0a 15 1c
page  35: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  37: 1b 01 13 01 16 0a 1b 1e 0f 1e 09 02 12 00 06 0c 12 04 01 1a 09 1d 1b 0b 0e 10 13 12 11 13 1d 0f
page  38: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b6 7f 7f 7f 7f 7f dc 7f 7f 7f 7f 7f 7f
page  39: 7f 7f 7f 97 7f 7f de 7f 7f 7f 7f 7f 7f 7f 7f 7f e6 d9 7f 7f 99 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  40: 00 0a 17 18 11 0e 10 0d 10 19 06 12 1c 1a 05 1d 1a 05 08 06 01 1e 0c 1b 03 00 1a 18 1e 15 10 17
page  41: 11 10 13 04 07 15 06 14 1d 0c 1d 13 1d 10 17 15 06 10 12 0e 14 1c 18 19 04 15 13 1d 00 05 1e 0b
page  42: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 94 7f 7f 7f 7f 7f 7f 80 c7 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  43: b7 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f db 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  44: 0d 0c 1c 04 00 1d 1b 1e 0d 1d 1c 06 17 19 14 10 08 0a 07 02 12 08 19 01 1c 15 1c 1b 1b 11 00 17
page  45: 1a 03 08 03 03 18 16 05 05 0c 17 19 17 12 04 0c 06 10 11 06 07 18 00 18 1b 1d 0b 11 12 13 1e 15
page  46: 16 08 06 07 06 12 13 16 03 17 0e 0b 0f 0d 06 0c 14 16 1c 06 1b 18 16 1e 04 18 1b 03 12 1d 09 1a
page  47: 13 02 00 19 08 07 1e 0e 19 0e 13 04 13 1a 10 16 14 01 17 12 09 00 1a 0e 16 1b 16 1c 0c 18 0d 1d
page  48: 03 17 0d 17 18 1e 14 1b 11 02 1b 02 02 19 08 14 00 06 16 01 0f 1d 06 15 17 13 17 15 00 05 10 1e
page  49: 19 09 08 1d 1d 09 0c 08 04 07 1e 02 07 06 02 10 17 19 13 19 00 08 1d 02 08 0e 08 10 01 07 1d 04
page  50: 0d 11 1e 10 18 09 16 12 17 1b 02 1d 12 08 0e 03 07 19 16 1a 05 03 0b 0f 16 09 16 12 08 18 02 12
page  51: 03 11 03 02 14 07 01 04 13 12 00 07 1d 06 11 0d 18 12 18 10 05 05 02 19 03 00 1d 06 1b 02 0e 06
page  52: 18 0d 17 0e 03 05 1a 0e 0b 0f 1a 0b 0b 0b 00 10 0e 0f 0d 0e 16 05 1e 03 1a 05 18 08 02 10 1a 1a
page  53: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ac 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  54: 1b 1d 0b 1b 18 08 0e 03 19 14 1b 18 14 16 11 03 12 00 04 18 01 02 03 1b 05 00 1a 03 1a 14 19 1d
page  55: 04 06 19 0d 09 0d 1c 07 00 1d 09 0b 1e 08 02 1b 07 06 1d 07 1b 0a 09 06 10 11 0c 0f 09 19 1c 17
page  56: 04 12 0a 0f 0b 09 0b 12 1e 1d 1a 19 08 1e 08 03 0f 16 0a 14 08 1d 0e 0e 10 1b 1e 10 0d 13 02 0d
page  57: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  58: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e5 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  59: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  61: 7f 7f 7f 7f e1 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ee 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  62: 7f 7f 7f 7f 7f 7f 7f ef 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  63: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  64: 03 10 1c 1a 0f 0c 02 08 09 1d 03 1d 0e 0d 08 1d 05 11 0f 06 06 1e 18 16 1b 03 15 17 06 0e 1e 0a
page  65: 0b 1a 17 0f 15 0d 18 07 10 12 0c 01 05 13 06 17 0f 1d 1a 16 0b 01 11 17 1c 06 04 1e 16 0f 16 04
```

```
page  66: 12 0b 1c 18 04 00 1d 0c 03 05 15 05 0b 14 0a 14 1d 18 13 13 09 13 14 12 0f 17 06 16 1e 01 07 1e
page  67: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d7 d4 7f 7f dd 7f 7f 7f 7f 7f 7f b1 7f 7f 7f 7f 7f 7f
page  68: 06 09 09 1b 02 0c 1a 11 07 1e 03 04 00 0d 1a 08 08 18 02 00 14 02 10 04 12 1a 0e 08 13 15 01 07
page  69: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 86 7f 7f 7f 7f
page  70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  71: 0c 0e 12 00 09 16 08 0e 07 0b 14 17 1d 0e 06 0a 01 07 12 12 07 05 03 05 0f 07 1b 11 0a 0d 01 16
page  72: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  73: 10 12 08 00 00 0a 06 11 08 17 12 14 16 10 0d 09 01 18 0f 03 1c 12 13 0d 03 1e 05 0b 1e 03 0f 0e
page  74: 02 13 00 1a 09 1c 1b 04 0c 04 17 02 02 03 1a 07 0a 0d 11 0c 1d 0a 0e 1a 07 05 19 09 1b 19 0b 0a
page  75: 03 1a 0d 10 14 04 16 18 1a 08 07 05 1b 0d 0c 10 15 09 0a 17 18 09 02 1c 16 12 0c 1d 0d 06 03 0a
page  76: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  77: 7f 7f 7f 7f 7f 7f 7f 7f 7f ad 7f 7f af 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  78: 17 1b 0a 12 0d 03 19 12 19 06 10 0e 16 02 0a 0f 02 11 16 0d 14 12 06 0a 1e 0a 0d 02 06 05 1c 16
page  79: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  81: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  82: 7f 7f 7f 7f 7f 7f 7f 89 7f 88 7f 7f 7f 7f 7f 7f eb ce 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  83: 7f 7f 7f 7f 7f 7f 7f f9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  84: 0c 1d 12 0a 07 1a 0e 18 0a 06 10 19 05 18 1c 18 19 00 13 1a 01 08 08 10 0d 0e 18 00 01 03 03 02
page  85: 0c 10 12 13 0d 01 18 1a 0f 11 08 1b 15 06 19 1e 0a 1e 0e 05 16 0a 16 12 03 10 1a 0e 10 1a 0d 0f
page  86: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  87: 1e 10 0c 0a 0e 18 0e 1d 04 09 10 0c 1a 19 1c 12 00 11 11 0f 08 15 1c 03 14 0b 0f 1b 1d 13 06 1c
page  88: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  89: 01 1c 1a 1c 15 13 1e 08 09 03 05 18 02 17 0c 1b 17 15 1a 01 08 0b 0d 14 1b 16 1c 15 0b 0f 17 19
page  90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  91: 19 1c 1a 0d 17 0f 03 01 02 06 04 0f 15 10 0d 14 09 0e 17 0c 05 1b 16 0b 0b 10 12 06 00 06 18 04
page  92: 1e 10 11 00 0c 11 00 13 13 01 13 0e 15 0a 15 16 00 01 14 1d 07 0e 12 09 0b 09 0b 12 09 0b 17 00
page  93: 19 11 1c 0b 0c 07 18 0e 08 05 16 12 16 0b 0f 04 16 00 0e 17 14 03 07 1a 13 1b 1b 0d 1b 16 0a 0b
page  94: 06 18 07 05 0d 15 03 09 0a 19 0d 1a 05 0a 14 1b 0d 06 03 10 05 19 19 05 08 19 13 18 0a 04 09 18
page  95: 02 1c 00 1e 07 19 19 1d 0e 15 09 19 03 1b 12 01 1d 09 02 08 00 08 16 1e 13 1a 19 16 0b 07 04 10
page  96: c2 7f 7f 7f 7f 7f 7f 7f 9c 7f 7f 7f 7f 7f 7f 7f 7f a9 7f 7f 7f 7f 7f 7f 7f 7f 7f d5 7f cb 7f
page  97: 08 06 16 19 14 06 16 10 0b 12 0d 14 17 1e 03 11 1d 14 11 1a 00 18 13 12 09 13 03 10 07 1c 18 0e
page  98: 7f 7f 7f c4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f a5 7f 7f 7f 7f 7f
page  99: 01 1a 0b 1e 0b 06 11 1b 0d 1a 16 0b 09 0f 0c 0b 14 1b 12 04 06 0b 13 04 02 09 08 00 10 1c 10 16
page 100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 101: 1b 0a 09 18 0d 1e 1a 1a 15 06 15 11 0e 11 09 06 1c 0b 03 1e 0f 14 0a 04 0a 19 08 1c 09 1b 05 0d
page 102: 01 17 0f 16 03 17 1c 01 0a 11 19 07 05 07 13 17 0c 0b 0c 0a 0c 02 0f 1e 0c 17 04 15 17 14 10 0e
page 103: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 104: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c9 7f 7f 7f 7f df 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f6 7f a2
page 105: 7f 7f 85 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 98 7f 7f 7f 7f 7f 7f 7f ea
page 106: 11 0e 16 1a 0c 16 1d 0e 07 07 16 14 1d 1a 07 05 08 05 15 1a 1b 07 1a 09 0d 16 02 02 19 09 0b 11
page 107: 1c 0d 15 03 1b 0b 0e 1b 08 05 19 12 02 00 0a 00 19 05 08 0c 0f 0d 13 14 0d 03 1e 15 02 0d 17 1e
page 108: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 109: 0f 09 09 0a 14 12 0b 05 0a 03 11 16 0b 02 05 0b 12 18 0b 18 13 0d 0b 0f 15 0d 15 0e 07 10 15 02
page 110: 12 16 00 17 14 0f 10 0e 05 10 01 0f 14 0d 11 1d 1b 04 18 13 01 0b 07 02 10 1c 0a 1a 15 04 1a 12
page 111: 0d 0a 05 07 1b 04 11 13 10 0f 0c 05 19 1b 11 07 1c 06 05 13 19 11 03 1e 04 0a 1e 18 17 02 1a 0c
page 112: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 113: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9d 7f 7f 7f 7f 7f
page 114: 02 17 1c 10 0d 16 05 08 06 12 09 07 15 1d 09 15 0c 1a 12 08 06 00 0e 0b 05 0b 09 18 04 1e 0e 12
page 115: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 116: 01 0d 09 06 1b 06 19 1d 03 1c 0c 06 05 01 0f 0b 1a 19 01 0c 0c 05 07 08 15 0a 03 06 0d 11 07 15
page 117: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 118: 0c 00 0e 00 15 1a 1c 01 14 09 0e 09 09 04 13 02 1d 01 1d 05 02 17 10 17 0f 13 02 14 0f 1d 00 02
page 119: 7f 7f 7f e3 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f fe
page 120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 121: 0e 14 12 13 0f 10 0f 19 05 18 09 1a 02 0f 0d 16 06 08 04 11 11 00 05 06 1d 0e 11 08 14 19 14 00
page 122: 7f 7f 7f 81 7f 7f 7f 7f 7f b3 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ca 7f 7f 7f 7f 7f 7f 7f 7f 7f b8 7f
page 123: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 124: 1b 0b 05 09 0e 10 10 0b 1a 08 0e 1b 19 09 07 19 00 04 10 10 05 01 06 17 0e 1e 18 1e 01 05 00 0d
page 125: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f fc 7f 7f a8 7f b2 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 126: 10 0c 1d 1b 1d 0f 17 0c 1e 1c 09 1c 05 02 16 09 10 13 01 17 08 0d 0a 17 17 08 03 09 0c 02 04 17
page 127: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```