CPE 453
“Operating Systems”

Final
March 19, 2004

• Time to complete the exam: 2 hours 50 minutes
• You will receive partial credit only if you show your work.
• Answer all questions in the space provided. (In case you need more space for a specific problem, use the back of that specific problem and make an appropriate note.)
• Be concise and clearly mark your answer.
• Put your (registered) name on EVERY page.
• GOOD LUCK!

Registered Name (Last, First):

Student ID#:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Put your name on every page</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 (15 points)

Fill in each blank with the letter that BEST describes the term. Letters are used no more than once.

1. _____ Process Control Block
2. _____ Interrupt
3. _____ Block device
4. _____ Critical Section
5. _____ Circular Wait
6. _____ i-node
7. _____ Quantum
8. _____ exec()
9. _____ Context-switch
10. _____ Compute-bound process
11. _____ A+
12. _____ Process
13. _____ TSL instruction
14. _____ XBOX
15. _____ Banker’s Algorithm

A. An algorithm used to steal money from banks.
B. Replaces the current process image with a new process image.
C. A console game system that is much better than a GameCube.
D. A hardware solution used to implement mutual exclusion.
E. The grade I want to get on the final.
F. A double western bacon cheeseburger with fries.
G. An asynchronous way of notifying the CPU that a device is “ready”.
H. A program in execution
I. A process that spends most of its time accessing I/O.
J. Device specific code used for controlling hardware devices.
K. The Unix process creation system call.
L. Occurs when the CPU saves the execution context of the current process and loads the execution context of a new process.
M. Provides random access to fixed-sized blocks.
N. An algorithm used to detect states that lead to deadlock.
O. A process that spends most of its quantum using the CPU.
P. A page replacement algorithm.
Q. A data structure used in a kernel to store information necessary for process management.
R. Cycle in processes holding and blocking on resources.
S. Don’t use me. I’m not the answer to any question.
T. Area in a program that can result in a race condition.
U. Time between timer interrupts.
V. Someone who sits on devices and tries to drive them around the block.
W. A filesystem format
X. A small data structure used to track and manage a file.

Problem 2 (15 points)

A. (6 points) In class, we discussed various memory management schemes that allow multiprogramming. Demand paging is the most popular scheme and is used in most computers. List three benefits of demand paging.

1. Processes can be bigger than main memory.
2. Processes do not have to be contiguous.
3. Reduces external fragmentation
4. Programmers deal with virtual memory, instead of managing physical memory
5. Increases multiprogramming

B. (4 points) What is a TLB (Translation Lookaside Buffer)?

*It is a small cache that holds the most recent (Virtual page no.) to (page frame no.) translation.*

C. (5 points) There are many tradeoffs one must consider when designing an operating system. One such tradeoff is that of page size. List the advantages and disadvantages of using a small page size.

(A) Reduce internal fragmentation (A) Less unused program in memory. (D) Large Page table. (D) Have to load more pages in memory.
Problem 3 \((15\text{ points})\)

A. \((5\text{ points})\) What is the difference between a physical address and a virtual address?

*Real memory uses physical addresses. These are the numbers that the memory chips react to on the bus. Virtual addresses are the logical addresses that refer to a process’ address space.*

B. \((6\text{ points})\) Use the LRU page replacement algorithm and the given reference string to calculate when page faults occur and distance string values. **SHOW ALL YOUR WORK.**

<table>
<thead>
<tr>
<th>Reference String</th>
<th>0</th>
<th>3</th>
<th>4</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>3</th>
<th>0</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Fault</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance String</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

C. \((4\text{ points})\) Would you recommend purchasing one more page of RAM? Why? \textit{No. According to the distance string, you would only eliminate one page fault by increasing the RAM size by 1.}
Problem 4 (15 points)

A. (4 points) Software that handles I/O is typically organized into four layers. List these layers.

(1) Interrupt Handlers (2) Device Drivers (3) Device Independent (4) User-level

B. (4 points) In which of the four I/O software layers is each of the following done.

1. Computing the track, sector, and head for a disk read. Device driver
2. Writing commands to the device register. Device driver
3. Checking to see if the user is permitted to use the device. Device Independent
4. Converting binary integers to ASCII for printing. User level

C. (7 points) Disk arm scheduling algorithms attempt to minimize the overhead of servicing data I/O requests on a disk drive. Say, 10 requests for I/O are distributed on the tracks of a hard drive as shown below. The requests are numbered in their order of arrival and placed in their corresponding tracks.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

The X denotes the initial track location of the disk arm.

Give the total amount of tracks the disk arm has to traverse (seek distance) in order to satisfy all the requests for the given disk arm scheduling algorithm: SHOW THE DISTANCE TRAVELED FOR EACH REQUEST.

1. First Come First Serve (FCFS):

seek distance=5+4+15+7+11+6+7+3+2+17=77

2. Elevator (The disk head is currently going left to right):

seek distance=1+2+2+1+2+1+14+2+3+1=29
Problem 5 (15 points)

In the Linux scheduler, the next process selected to run is based on the “goodness” of the process. Below is the “goodness” function found in the Linux 2.4.20 kernel. Note: I only show code dealing with non-real-time processes on a single processor.

```c
static inline int goodness(struct task_struct * p, int this_cpu, struct mm_struct *this_mm)
{
    int weight;
    weight = p->counter;
    if( !weight )
        goto out;
    weight += 20 - p->nice;
out:
    return weight;
}
```

Suppose we change the “goodness” function to be the following:

```c
static inline int goodness(struct task_struct * p, int this_cpu, struct mm_struct *this_mm)
{
    int weight;
    weight = p->counter;
    return weight;
}
```

Discuss all the implications of using the new “goodness” function.
Problem 6 (10 points)

Given the outstanding instruction provided to you in your operating systems course at CalPoly, you have been hired by Initech to add a new system call to the Linux kernel. The new system call allows the programmer to set how many disk blocks will be fetched in the Readahead file system optimization implemented in Linux. In the Readahead optimization, everytime the OS needs to access the disk to read a block, it reads additional blocks to potentially handle future reads from the disk block cache.

A. (4 points) List the major steps you need to do to implement this new system call. Think about the steps you took in Lab 4.

1. Add system call number to system call table
2. Add .h file that associates a function with the system call number.
3. Write system call
4. Change Makefile
5. compile
6. boot to new kernel

C. (3 points) Describe one major advantage of the new system call.

Prefetching optimized on a per process basis.

D. (3 points) Describe one major disadvantage of the new system call.

Users may unfairly monopolize the disk buffer cache.
Problem 7 (20 points)

A semaphore is a new variable type that counts the number of available resources. There are two indivisible actions performed on a semaphore. They are \texttt{down()} and \texttt{up()}. In lab 2, we used a \texttt{mutex}, but not a true counting semaphore. In this problem, you will use a POSIX \texttt{semaphore} to complete the code given on the next page to prevent buffer overflow and buffer underflow. You will also need to use a \texttt{mutex} or \texttt{sempahore} to prevent any race conditions.

Here are some function prototypes contained in \texttt{pthread.h} and \texttt{semaphore.h} you may find useful:

- \texttt{int sem\_init(sem\_t \*sem, int pshared, unsigned int value)}
- \texttt{int sem\_wait(sem\_t \*sem)}
- \texttt{int sem\_post(sem\_t \*sem)}
- \texttt{int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex)}
- \texttt{int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex)}

Semaphore operation notes:

1. The man page for all semaphore operations are attached to this problem.
2. Use \texttt{sem\_init()} to assign initial values to your semaphores.
3. \texttt{sem\_wait()} is equivalent to the \texttt{down()} operation.
4. \texttt{sem\_post()} is equivalent to the \texttt{up()} operation.

Grading Notes:

1. Write your code in C.
2. Assume all your code is contained in a single “.c” file.
3. I will be grading for correct C usage and syntax as well. In other words, I should be able to type in your program and compile it without error.
4. You don’t have to include error checking code.

Fill in the missing code (see commented sections indicating Parts A-E) for the program given on the next page.
#include<pthread.h>
#include<semaphore.h>
#define BUFFSIZ 50
#define LOOP 50000

sem_t aAmount, bAmount; /* Semaphores */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void addb(void); void adda(void); /* function prototypes */
char buff[BUFFSIZ]; int i=0; /* Shared variables */

int main() {
    pthread_t t1, t2;
    memset(buff, 'b', BUFFSIZ);
    /* PART A: Initialize semaphores */
}

void adda() {
    int cnt=LOOP;
    while( cnt-- ) {
        /* PART B: prevent buffer overflow and race conditions */
        sem_wait(&bAmount); pthread_mutex_lock(&mutex);
        buff[i++] = 'a';
        /* PART C: prevent buffer overflow and race conditions */
        pthread_mutex_unlock(&mutex); sem_post(&aAmount);
    }
}

void addb() {
    int cnt=LOOP;
    while( cnt-- ) {
        /* PART D: prevent buffer underflow and race conditions */
        sem_wait(&aAmount); pthread_mutex_lock(&mutex);
        buff[--i] = 'b';
        /* PART E: prevent buffer underflow and race conditions */
        pthread_mutex_unlock(&mutex); sem_post(&bAmount);
    }
}