CIS 415: Operating Systems

Prof. Allen D. Malony

Midterm – May 7, 2019

NAME: _____

Section	Concepts	Problems	Total	Score
1. Processes and Threads	18	15	33	
2. CPU Scheduling	18	15	33	
3. Synchronization / Deadlocks	18	25	43	
Total	54	55	109	

Comments:

1. Take a deep breath. Congratulations on studying hard!
2. Write your name on the front page now and initial all other pages.
3. You have 80 minutes. There are 3 sections. Do all sections. Each section is designed to take 25 minutes, more or less.
4. Concept questions are approximately 50% of the total points. They are intended to be short answer. If you are spending more than 5 minutes on ANY concept questions, you are writing too much! Move on!
5. If you have a question, come down to the front.

Exams should be challenging learning experiences. Enjoy it!!!

1 Processes and Threads (33)

1.1 Concepts (18) (CHOOSE ONLY 3 TO ANSWER !!!)

What are the possible states of a process during its execution? What are the reasons for state transitions? Draw a diagram to help explain your answer.

What is the process address space? What is in it and how are things structured (i.e., where are things located)? Draw a picture to show this. Describe what would be different if there were multiple threads (e.g., 2 threads).

What is the purpose of interrupts? How does an interrupt differ from a trap? Can traps be generated intentionally by a user program? If so, for what purpose?

What is interprocess communication (IPC)? What are 2 IPC models? Name 2 types of IPC?

Describe the actions taken by a kernel to context-switch between kernel-level threads?

What is the difference between the 1:1 and M:1 threading models? Why might we prefer a 1:1 threading model?

1.2 Problems (15) (CHOOSE ONLY 1 TO ANSWER !!!)

1.2.1 Processes Galore! (15)

Consider the following program. Assuming all `fork()` calls are successful, how many processes are created? Draw the process hierarchy at the time all processes are sleeping. (You can assume that all of the processes that are created will all be sleeping before any process returns.) At that time, what are the values of the variables: `pid1`, `pid2`, `pid3`, `pid4`? If you need to know a Unix process identifier, just make up a unique 5-digit number.

```
#include <stdio.h>
#include <unistd.h>

pid_t pid1, pid2, pid3, pid4;

int main()
{
  pid1 = -1; pid2 = -2; pid3 = -3; pid4 = -4;
  pid2 = fork();  // 2 processes running after this
  pid3 = fork();  // 4 processes running after this
  pid4 = fork();  // 8 processes running after this
  sleep(10); // sleep for 10 seconds
  return 0;
}
```

1.2.2 What is the value of this problem? (15)

Consider the following program which uses the Pthreads API.

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param);

int main(int argc, char *argv[])
{
pid_t          pid;
pthread_t      tid;
pthread_attr_t attr;

  pid = fork();
  if (pid == 0) {
    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, NULL);
    /* LINE A */
    /* value = 10; */
    pthread_join(tid, NULL);
    printf("CHILD: value = %d", value);  /* LINE C */
  }
  else if (pid > 0) {
    wait(NULL);
    printf{"PARENT: value = %d", value); /* LINE P */
  }
}

void *runner(void *param) {
  value = 5;
  pthread_exit(0);
}
```

a) What would be the output from the program at `LINE C` and `LINE P`?

b) If the statement below `LINE A` is uncommented, what would be your answer to a)?

2 Scheduling (33)

2.1 Concepts (18) (CHOOSE ONLY 3 TO ANSWER !!!)

What is the *CPU-I/O burst cycle* and why is it relevant to processor scheduling?

What is the different between turnaround time and response time with respect to CPU scheduling?

What is the quantum in Round-Robin scheduling? How does it get implemented? What happens in Round-Robin scheduling as the quantum increases? What happens in Round-Robin scheduling as the quantum decreases?

What is the idea behind using a multilevel queueing system for scheduling? What advantage is there in having different quantum sizes at different levels of a multilevel queueing system?

Under what condition is *shortest job first* scheduling optimal with respect to minimizing the average waiting time? Explain briefly.

Suppose there are multiple processors in the system. Does it make sense to use different scheduling algorithms for each processors? Explain your answer.

2.2 Problems (15) (CHOOSE ONLY 1 TO ANSWER !!!)

2.2.1 Changing Priorities (15)

Priority-based process scheduling selects the next process to be allocated to the CPU based on the process's priority, computed as a function of process and system parameters. The behavior of most scheduling algorithms can be replicated by a priority-based scheduler with the proper choice of priority function. For each of the scheduling algorithms below:

i. Give a brief definition of what the scheduling algorithm does.

ii. Specify the priority function as a simple function (arithmetic operators) of one or more of the following parameters:

a = attained service time

r = real time the process has spent in the system

t = total service time required by the process

The process with the HIGHEST priority is scheduled next.

FIFO (First In / First Out)

LIFO (Last In / First Out)

SJF (Shortest Job First)

RR (Round Robin)

SRT (Shortest Remaining Time)

2.2.2 Just Take it Easy, We Will Get to You Eventually (15)

Per Brinch Hansen was a famous computer scientist working in the field of concurrent programming and operating systems. He developed an interesting scheduling strategy called *Highest-Response-ratio-Next (HRN)* that corrects some of the weaknesses in the shortest-job-first (SJF) strategy. HRN is a nonpreemptive scheduling discipline in which the priority of each job is a function not only of the job's service (run) time, but also of the amount of time the job has been waiting for service. Priorities in HRN are calculated dynamically according to the formula

$$priority = \frac{timewaiting + servicetime}{servicetime}$$

Is indefinite postponement a problem with SJF? How does HRN prevent it?

How does HRN decrease the favoritism shown by other strategies to short new jobs?

Suppose two jobs have been waiting for about the same time. Are their priorities about the same? Explain your answer.

3 Synchronization / Deadlocks (43)

3.1 Concepts (18) (CHOOSE ONLY 3 TO ANSWER !!!)

Explain the concept of concurrency. Why do we care about concurrency in operating systems? Is it advantageous to have logical concurrency even without physical concurrency (e.g., multiple CPUs)? Give one reason why.

What does it mean for a set of instructions to be atomic?

Consider the following twist on implementing a blocking semaphore. Suppose that the `wait()` routine accepts a boolean variable `block` which will cause blocking to occur if `block = 1` and the wait is unsuccessful. However, if `block = 0` and the wait is unsuccessful, no blocking will occur, and the `wait()` will return a 0 to indicate that the wait was unsuccessful. Otherwise, `wait()` will return a 1. Why might a process want to use this "maybe" blocking semaphore?

What are the necessary conditions for a deadlock to occur?

What is the difference between deadlock prevention and deadlock avoidance?

What is a race condition?

3.2 Problems (25) (DO BOTH PROBLEMS !!!)

3.2.1 "Pipelined" Producer-Consumer with Bounded Buffers (15)

Suppose two threads share a common fixed-size buffer. One thread, the *producer*, puts information into the buffer, and the other thread, the *consumer*, takes it out. The following code shows what is done when the producer "produces" an item and when the consumer "consumes" an item. Assume that the producer will call `produce()` repeatedly and likewise the consumer will call `consume()` repeatedly. Is the code correct? If not, why not and how would you fix it?

```
#define N 100
int count = 0;               /* number of items in buffer   */

void producer(void) {
  int item;

  produce_item(&item);     /* generate next item          */
  while (count == N) ;     /* if buffer is full, wait     */
  enterItem(item);         /* put item in buffer          */
  count = count + 1;       /* increment buffer item count */

} /* producer */

void consumer(void) {
  int item;

  while (count == 0);      /* if buffer is empty, wait    */
  removeItem(&item);       /* take item out of buffer     */
  count = count - 1;       /* decrement buffer item count */
  consume_item(item);      /* process item                */

} /* consumer */
```

The procedures `enter_item()` and `remove_item()`, not shown, handle the bookkeeping of putting items into and taking items out of the buffer.

Now consider a stream of items produced by a thread T_0 which then pass through a sequence of threads, T_1, T_2, ..., T_{m-1}, where T_i consumes items from a shared buffer with T_{i-1} and produces updated items into a shared buffer with T_{i+1}, as depicted below:

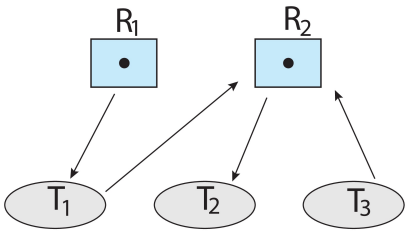
$$T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_{m-1}$$

Thread T_0 only produces and T_{m-1} only consumes. Rewrite the code (you one consider to be correct) from above to work in this *pipeline* scenario. Assume that all buffers have the same length, N. Show what a generic thread T_i does. Assume that all threads run forever in their respective roles. Explain your strategy. Do you see any problems? In particular, is it possible to deadlock?

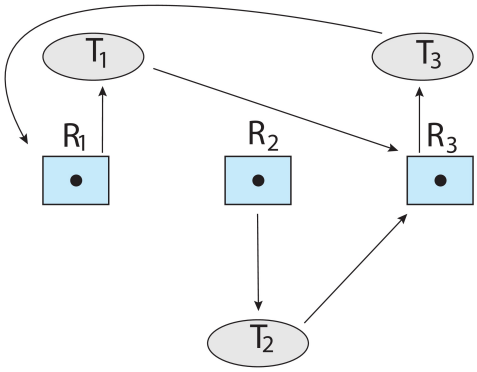
3.2.2 To be deadlocked or Not to be deadlocked, That is the question! (10)

Which of the six resource allocation graphs shown below illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where this is not a deadlock situation, illustrate the order in which the threads may complete execution.

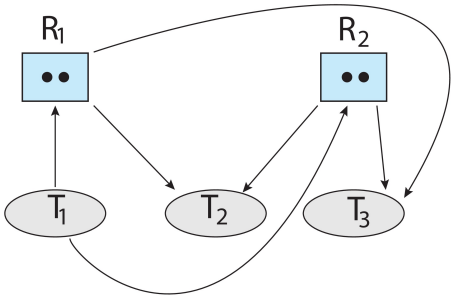
(a)



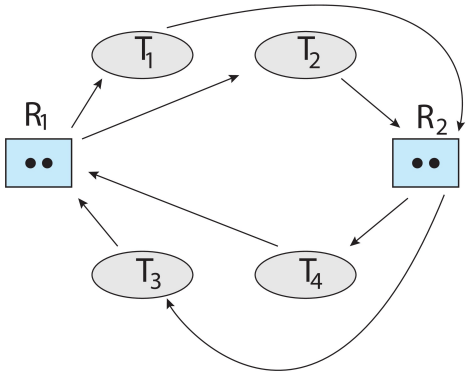
(b)



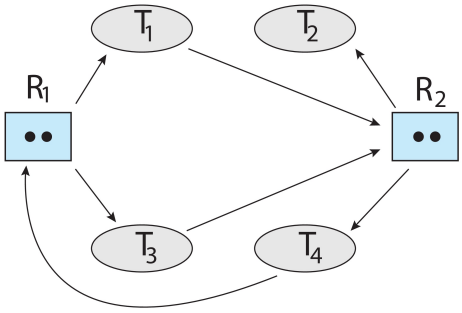
(c)



(d)



(e)



(f)