COMP 252 MISC

27 OCT 1997

COMP 252 Principles of Systems Software

~~Lecture Section 1~~

Fall Semester 1997

Solutions : **Midterm Examination**

Date: October 13, 1997 Time: 11:00 - 11:50

Name: Solution	Student ID:	Email:

Instructions:

1. This examination paper consists of 5 pages and 5 questions.
2. Please write your name, student ID and Email on this page.
3. For each subsequent page, please write your student ID at the top of the page in the space provided.
4. Please answer all the questions within the space provided on the examination paper. You may use the back of the pages for your rough work.
5. Please read *each question very carefully* and answer the question clearly and to the point. Make sure that your answers are neatly written, readable and legible.
6. Show all the steps you use in deriving your answer, wherever appropriate.
7. For each of the questions assume that the concepts are known to the graders. Concentrate on answering to the point what is asked. Do not define or describe the concepts.

Question	Points	Score
1	20	
2	10	
3	15	
4	25	
5	30	
TOTAL	100	

1. Answer the following true/false questions by circling either T or F. (20 points)

a) If no thread/process is waiting on a condition variable, then a signal operation executed on the condition variable by a thread will have no effect. (T) F

b) The operating system executes in user mode. T (F)

c) When a timer interrupt occurs, the currently executing process/thread on the CPU will be blocked and moved to waiting state. T (F)

d) A child process in UNIX shares its address space with its parent process. T (F)

e) A multithreaded process has only one program counter value associated with its address space. T (F)

f) In UNIX an execve() system call results in the creation of a new address space. T (F)

g) In a multithreaded process, a thread does not share its data section with any of the other threads. T (F)

h) An operating system is interrupt driven. (T) F

i) Multiprogramming does not require the support of any memory protection mechanism. T (F)

j) Buffering permits the overlap of the I/O of one job with the execution of another job. T (F)

2. Describe (in no more than five sentences) the steps involved in creating and initializing a thread (e.g., in nachos, this is done in two parts: thread::thread() and thread::fork()). (10 points)

2 Allocate a Thread Control Block (TCB). Within the TCB,
1 initialize IP to point to the correct code,
1 DP to point to global dataspace.

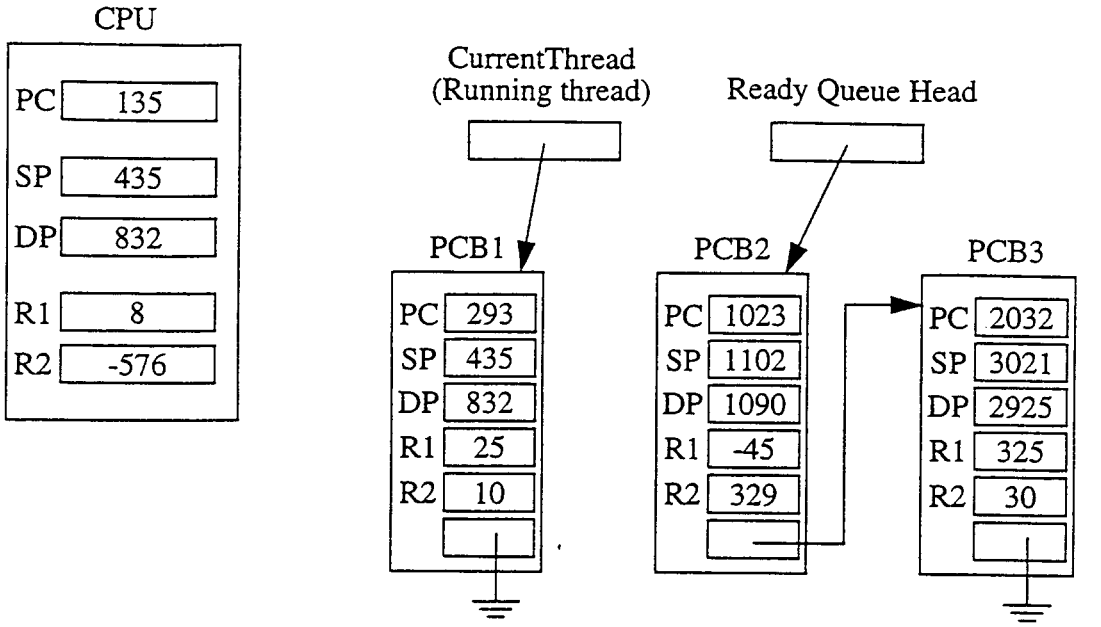
2 Allocate an execution stack space for the thread.

2 Initialize the SP (inside the TCB) to point to this stack space.

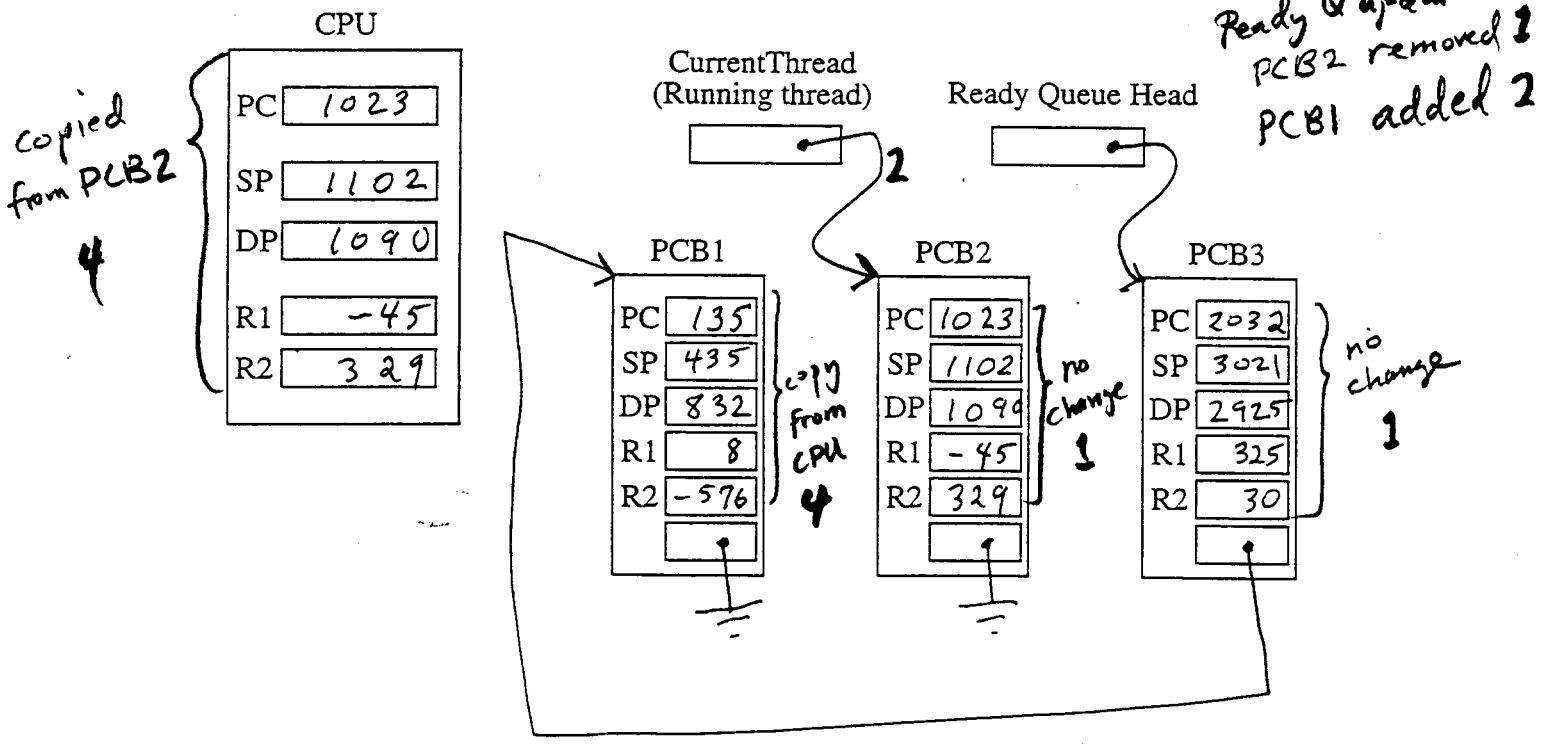
2 Place the TCB on the ready-to-run list.

3. The current state of the computer system is shown in the figure below. If a context switch occurs at this point, what will be the state of the computer system after the completion of the context switch? Assume that the next process to be allocated the CPU is the process at the head of the ready queue, and that the context switch is caused by a timer interrupt. (20 points)

   a) Before Context Switch:



   b) After Context Switch:



4. Three threads T1, T2 and T3 want to execute within a critical section. We want to impose a strict ordering of execution of these threads (25 points)

   a) First T1 should execute within the critical section (CS),

   b) when T1 finishes its execution in the CS then T2 will be allowed to execute in the CS,

   c) when T2 finishes in the CS, then T3 will be allowed to execute in the CS, and

   d) when T3 finishes in the CS, then T1 is allowed to execute the CS next, and so on.

   Thus the order of execution of the three threads within the critical section is T1, T2, T3, T1, T2, T3, ... (hint: think about synchronization using semaphores).

   a) How many semaphores will be required to achieve this ordering, and what should be their initial values?

3 semaphores are needed
S1 with initial value 1
S2 " " 0
S3 " " 0

   b) Sketch the code for the three threads and show how you can use the semaphore P() and V() operations to enforce the ordering..

T1:
```
while (TRUE){
    S1 -> P()
    CS
    S2 -> V()
}
```

T2:
```
while (TRUE){
    S2 -> P()
    CS
    S3 -> V()
}
```

T3:
```
while (TRUE){
    S3 -> P()
    CS
    S1 -> V()
}
```

This solution is worth 25 points

- The following solution with 4 semaphores, S1, S2, S3 and mutex is worth only 20 points since mutual exclusion of T1, T2 and T3 is already provided by S1, S2, S3

```
T1: while (TRUE){
    S1 -> P()
    mutex -> P()
    CS
    S2 -> V()
    mutex -> V()
}
```

5. The operating system maintains a single global printer queue from which a printer controller process picks up the next print job to be printed. Processes requiring use of the printer will queue up their print job into the global printer queue. When there is no printing to be done, the printer is idle. Implement a mechanism to coordinate access to the *global shared* printer queue among the printer controller process and other processes. A rough skeleton of the printer controller process and a user process is given below for your convenience. You may use either semaphores or locks/condition variables. You may also consider the printer queue size to be unbounded. (30 points)

```
List *printer_queue;

printer_controller () {
    while (TRUE) {
        // if no job in queue, then idle
        ...
        printer_queue->Remove();
        ...
        ...
        Print the job on the printer
    }
}
```

```
user_process() {

    Do other work;
    ...
    printer_queue->Add();
    // if printer idle, wake it up
    ...
    Do other work

}
```

Semaphore mutex = 1;
Semaphore controller = 0;

controller->P();
mutex->P();
mutex->V();
mutex->P();
mutex->V(
controller->V()

1. Correct number of semaphores and correct initial values. 10 points
2. Proper mutual exclusion on the queue 10 points
3. Proper Synchronization (wakeup of idle printer by user process) 10 points.