University ofOttawa
School of Information Technology and Engineering

Course:	CSI3310 – Operating Systems Principles	**Professor:**	Stefan Dobrev
SEMESTER:	Winter 2007	**Date:**	February 26 2007
		Hour:	14:30-16:00 (1.5 hours)
		Room:	Montpetit 202

Midterm Exam
Solution

The exam consists of three (3) parts:

Part 1	**Multiple Choice**	**8 points**
Part 2	**Short Answers**	**10 points**
Part 3	**Problem Solving**	**10 points**
Total		**25+3 points**

The exam is worth 28 points, 3 of them are essentially bonus (25 points represent 25% of the total mark).

Part 1: Multiple choice:

1. b 2. d 3. b,e 4. c 5. d 6. a, d 7. a 8. c

Part 2: Short answer questions

Question 1 :

Symmetric multiprocessing: All CPU can perform scheduling; partitioning of tasks; and all OS functions.
Asymmetric multiprocessing: These functions are allocated to a single CPU; the other CPUs are dedicated to executing user processes.

Question 2;

Microkernel structure:

- The kernel of the first OSs were monolithique that contained all OS functions.
- Later an effort was made to include only essential functions in a smaller kernel (e.g. scheduling and message passing), and use system processes for other OS functions (e.g. file systems).

Modular structure :

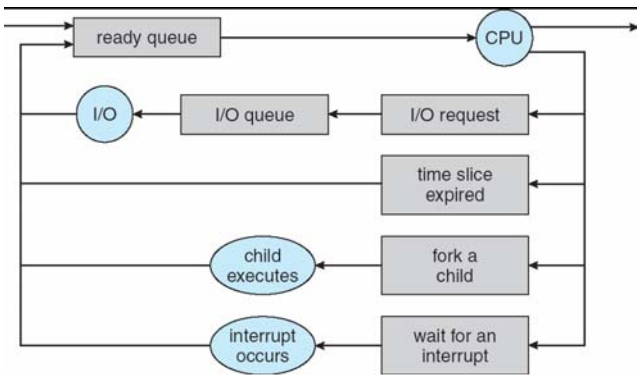
- Many modern OSs used an approach that uses modules :
  - Object oriented approach
  - Each component is separate
  - Standard interfaces are used to access functions
  - Modules are dynamically loaded by the kernel according to system needs.
- The structure is similar to the layered structure, but much more flexible.

Question 3 :

- Cancellation refers to termination of a thread requested by another thread. In the case of deferred cancellation, a flag is set to indicate termination; the thread verifies the flag to see if it should terminate its execution. This allows the thread to clean up its resources. This approach gives a graceful termination.

Question 4 :

The PCBs are placed in the different queues according to their state. For example, PCBs whose processes are in the Ready state are placed in the Ready queue. PCBs of processes in the wait state are placed in the I/O queues. CPU scheduling selects a process in the Ready queue for execution (allocated to the CPU). Note that the Ready queue and the I/O queues are typically implemented as multiple queues.



Question 5 :

No, it is not valid. It does note satisfy the progress requirement, since each process must wait for the other process to pas through its CS, even if the other process is in its remainder section.

Question 6 :

```
wait(S):    S.value --;
            if S.value < 0
            { // Semaphore is busy if value < 0
                     Add the process/thread to S.L;
                  block() // the process/thread is placed in the wait state
            }
```

The wait() is used to bloc a process until a signal is received from another process.

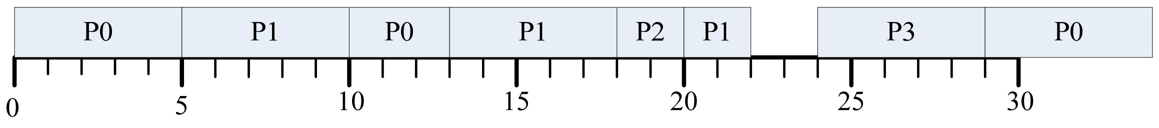
```
signal(S): S.value ++;
      if S.value ≤ 0
      {  // Processes/threads are waiting in the queue if value ≤ 0
              Remove a process/thread P from S.L;
          wakeup(P)  // selected process/thread becomes ready
      }
```

The signal() call is used to allow blocked processes to continue their execution.

Part 3: Problem Solving

Question 1 :

Time	Wait	Ready	CPU
0			P0(8), 5
2		P1(12)	P0(6), 3
5		P0(3)	P1(12), 5
10		P1(7)	P0(3), 5
13	P0(12)		P1(7), 5
18	P0(7)	P1(2)	P2(2), 5
20	P2(12), P0(5)		P1(2), 5
22	P1(12), P2(10), P0(3)		
24	P1(10), P2(8), P0(1)		P3(6), 5
25	P1(9), P2(7)	P0(9)	P3(5), 4
29	P3(12), P1(5), P2(3)		P0(9), 5
32	P3(9), P1(2)	P2(4)	P0(6), 2



Question 2:

```
int main(int argc, char *argv[])
{

   int i,j;             /*indexes into arrays */
   char *pgrm1;         /*pointer to first program */
   char *pgrm2;         /*pointer to second program */
   int pipe1to2[2];  /* pipe to prc1 to prc2 */
   int pipe2to1[2];  /* pipe to prc2 to prc1 */
   int pid;

   if(argc != 3)
   {
     printf("Usage: stdout2stdin <pgrm1> <pgrm2> \n");
     exit(1);
   }

   /* get programs */
   pgrm1 = argv[1];
   pgrm2 = argv[2];

   /* create the pipes */
   pipe(pipe1to2);
   pipe(pipe2to1);
   /* create process 1 */
   pid = fork();
   if(pid == 0)
   {
      dup2(pipe1to2[0], 0);
      dup2(pipe2to1[1], 1);
      close(pipe1to2[0]);
      close(pipe1to2[1]);
      close(pipe2to1[0]);
      close(pipe2to1[1]);
      execlp(pgrm1, pgrm1, NULL);
   }

   /* create process 2 */
   pid = fork();
   if(pid == 0)
   {
      dup2(pipe2to1[0], 0);
      dup2(pipe1to2[1], 1);
      close(pipe1to2[0]);
      close(pipe1to2[1]);
      close(pipe2to1[0]);
      close(pipe2to1[1]);
      execlp(pgrm2, pgrm2, NULL);
   }
}
```