York University
Faculty of Science and Engineering
Department of Computer Science and Engineering

Midterm
CSE 3221 Z Operating Systems Fundamentals
Feb. 15, 2011 (2:30-4:00pm)

Section: ____________________Z_____________________________________

Family Name: ____

Given Name: ____

CS Account: ____

Student Id: ___________________SOLUTION_______________________________

Instructions
1. The exam has 6 questions and 10 pages (including this one).
2. No aids permitted.
3. Answer each question in the space provided. If you need more space write on the backs of pages.
4. **Examination time is 90 minutes.**
5. Answers written in pencil or erasable ink will *not* be considered for remarking.
6. Do *not* use red ink. Write legibly. Unreadable answers do *not* count.
7. **No questions re the interpretation, intention, etc. of an exam question will be answered by invigilators. If in doubt, state your interpretation as part of your answer.**

Question	Maximum	Mark
1	8	
2	8	
3	14	
4	10	
5	10	
6	10	
Total	60	

1. (8 marks) Interrupt handlers are a critical component of OS kernel. Please complete the following description about how an interrupt is handled in a sequential vectored interrupt system.

MARKING SCHEME: deduct one mark for each wrong or blank spot.

I) CPU normally executes instructions one by one until an interrupt happens. Give two examples of what events may cause an interrupt:

(I.1) [**any hardware events, such as a stroke in keyboard, completion of I/O event, hardware failure, timer, I/O device and so on**]

(I.2) [**software interrupt such as TRAP instruction**]

II) When an interrupt happens:

(II.1) ___ **Hardware** ______ (Hardware or OS kernel) switches CPU to ___**kernel /system/monitor**_____ mode.

(II.2) Load PC register according to ____**interrupt vectors**_____ and jumps to run an interrupt handler.

III) An interrupt handler typically executes the following steps:

III. 1) ________ **disable** _________________ (enable or disable) interrupt.

III. 2) Save ________**CPU status or CPU context**_________________.

III. 3) Deal with the interrupt.

III. 4) Restore ______________ **CPU status or CPU context** _________.

III. 5) ____**OS kernel switches CPU to user mode**___________________.

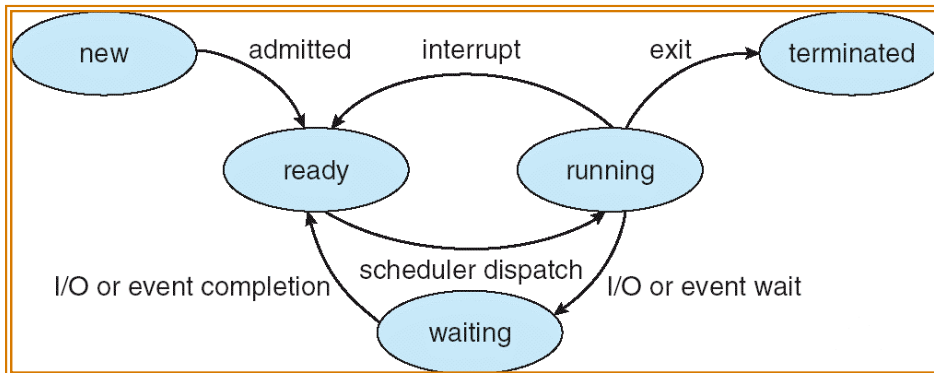
III. 6) ________ **enable** _________________ (enable or disable) interrupt.

III. 7) return from the interrupt handler (restore PC register) .

IV) CPU continues to execute next instruction.

2. (8 marks) Draw a diagram of the five-state process model, showing all states and transitions. Label each transition with all possible events that may cause such a transition.

Diagram:



Marking schemes:

*) Diagram (4 marks): label each state correctly; draw all transition correctly, deduct point for any extra transition.

*) Label transitions (4 marks).

3. (14 marks) Indicate whether each of the following statements is **true** or **false**. Write the entire word ``**true**'' or ``**false**'' in the box after each statement. **One (1) point for each correct answer; 0 for no answer; No mark deduction for wrong answers.**

(3.1) Non-preemptive CPU scheduler does not allow context switch in middle of CPU bursts.

[TRUE]

(3.2) All system calls are implemented as part of an interrupt handler in kernel space.

[TRUE]

(3.3) DMA controller uses an interrupt to inform OS that it has finished data transmission.

[TRUE]

(3.4) An reentrant program can not make any system calls.

[FALSE]

(3.5) It is faster to switch context between two threads from different processes than two threads belonging to one process.

[FALSE]

(3.6) Thread library runs in kernel space since it needs to map user thread to a kernel thread.

[FALSE]

(3.7) In multiprogramming system, a process is always running in its CPU burst.

[FALSE]

(3.8) Multiple threads that are part of the same process have their own program counters and stack pointer values. [TRUE]

(3.9) Multiple threads that are part of the same process have their own copies of global variables.
[FALSE]

(3.10) Multiple threads that are part of the same process share all open files.
[TRUE]

(3.11) Multiple threads that are part of the same process have their own copies of local variables.
[TRUE]

(3.12) Contex switch is alwayes triggered by an interrupt.
[TRUE]

(3.13) Unix signals can be directly sent from one process to another process without kernel.
[FALSE]

(3.14) A multiprogramming OS needs to have multiple CPUs to run several processes concurrently. [FALSE]

4. (10 marks) Short questions:

4.1) (2 marks each) In round robin scheduling,

(a) What is one advantage of having a long time quantum?

Small number of context switch.

(b) What is one advantage of having a shorter time quantum?

Ensure fairness among all processes.

(c) What scheduling algorithm is approximated with a very long quantum?

FCFS (first come first serve)

4.2) (1 mark each) In a dual-mode CPU architecture, specify which category (**normal** or **previledged**) each of the following instructions belongs to:

(d) TRAP instruction [**normal**]

(e) Turn off interrupt [**previledged**]

(f) Add two general registers and save the result to memory [**normal**]

(g) Read status of an I/O port [**previledged**]

5. (10 marks) Please describe all possible outputs for the following two programs when they run **normally** in a Unix system. Also clearly explain how and why each output happens.

5.1) (5 marks) Program A:

```
#include <stdio.h>
int num = 0 ;
int main(int argc, char *argv[])
{
  int pid ;

  pid = fork() ;

  printf("%d",num) ;

  if (pid == 0) {
    num = 1;
  } else if (pid > 0) {
    num = 2 ;
  }
  printf("%d",num) ;
}
```

0102 or

0012 or

0201 or

0021

***) execution order of two processes is unexpected, up to CPU scheduler.**

***) pissible execution interleaving.**

5.2) (5 marks) Program B:

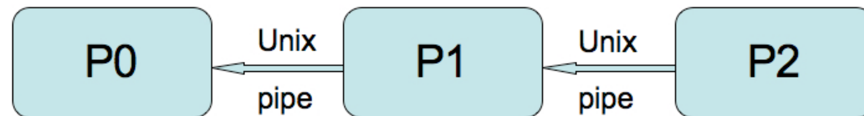
```
#include <pthread.h>
#include <stdio.h>
int X = 0, Y = 0 ;
void *runner1()
{
  for(; X<4; X++) {
   Y = 0 ;
   printf("%d",X) ;
   Y = 1;
  }
}
void *runner2()
{
  while (X<4) {
     if ( Y == 1) printf("a") ;
   }
}
int main(int argc, char *argv[])
{
   pthread_t tid1, tid2; /* the thread identifier */
    /* create the thread 1 & 2*/
   pthread_create(&tid2,NULL,runner2,NULL);
   pthread_create(&tid1,NULL,runner1,NULL);
   pthread_join(tid2,NULL) ;
   pthread_join(tid1,NULL) ;
}
```

0 a … a 1 a … a 2 a … a 3 a … a

1) interleaving of two threads due to unknown execution order

2) thread1 prints 0, 1, 2, 3 in order; thread2 prints a number of a only when thread1 turns Y to 1.

6. (10 marks) Write a C program to implement the following three processes in a Unix system:



where `P0` is parent of `P1` and `P1` is parent of `P2`. Two Unix pipes are created from child proces to parent process as above. In your program, `P2` sends its pid to `P1` through the pipe; `P1` is waiting to read whatever message from this pipe and passes it to `P0` through another pipe; `P0` is waiting to read the message and print onto the screen.

System calls for reference:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);

ssize_t write(int fd, const void *buf, size_t count);

int close(int fd);

int pipe( int filedes[2] ) ;

pid_t fork(void);

pid_t getpid(void);
```

```
#include <unistd.h>
#include <stdlib.h>
int main() {
  int n, fd1[2], fd2[2];
  pid_t  pid ;
  long int num ;

  if( pipe(fd1) < 0 )   err_sys("pipe1 error") ;
  if ( (pid = fork()) < 0 ) err_sys("fork error") ;
  else if ( pid > 0 )  { /* parent P0 */
      pid_t num ;
      close(fd1[1]) ;
      read(fd1[0], &num, sizeof(num)) ;
      print("%d\n", num) ;
 } else  {     /* child process P1 */
   close(fd1[0]) ;
   if (pipe(fd2) <0)  err_sys("pipe2 error") ; ;
   if ( (pid = fork()) < 0)  err_sys("fork error") ;
   else if ( pid == 0 )  {  /* grand-child process P2 */
      close(fd2[0]) ;
      pid = getpid(void);
      write(fd2[1], &pid, sizeof(pid)) ;
   } else {  /* Process P1 */
      pid_t num;
      close(fd2[1]) ;
      while (read(fd2[0], &num, sizeof(num))==0) ;
      write(fd1[1], &num, sizeof(num) ) ;
  }
```