Com S 352 Exam 1

Friday, September 29, 2017

Time: 50 minutes

Last Name: ____Solutions_____
First Name: _____

The exam requires books, notes and electronic devices to be closed. Please go over all the questions in the exam before you start working on it, and attempt the questions that seem easier first. If you see yourself getting stuck on one question, continue on and come back to it later if you have time. We are looking for brief but to-the-point correct answers to all questions. Good luck!

I. TRUE/FALSE questions. [2pts * 5 = 10pts]

__F___ The operating system switches kernel threads by switching thread IDs.

___F__ A Shell converts all user inputs into system calls.

__F___Processes all share the system stack when they are in user mode.

___F___ In Symmetric Multi-Processor (SMP) Scheduling, soft affinity does not allow a process to migrate between processors during its execution.

___F___ It is more expensive (in terms of resources used) for new thread creation than new process creation.

Ⅰ. Multiple choices questions. When there are multiple correct answers, you need to select all correct answers to get full credits. Points will be deducted for wrong answers. [4pts * 5 = 20pts]

1. Which of the following process state transitions can cause a non-preemptive scheduler to swap in a process to run on the CPU? _____

   (A) A process changes from NEW to READY.
   (B) A process changes from RUNNING to WAITING.
   (C) A process changes from RUNNING to TERMINATED.
   (D) A process changes from WAITING to READY.

2. Which of the following is (are) true about the execution of "printf(``Hello!'');" in a Linux system? _____

   (A) A function of the C library is called.
   (B) A system call is made to the kernel.
   (C) The execution mode switches from "user" to "kernel".
   (D) The execution mode switches from "kernel" to "user".

3. Which of the following is (are) true about user-level threads? _____

   (A) A user thread is not created by the OS kernel.
   (B) A user thread is scheduled by the OS kernel.
   (C) A user thread cannot make a system call.
   (D) Context of a user thread is stored in the Process Control Block (PCB).

4. Which of the following is (are) true about a multi-thread process? _____

   (A) All the threads within the process share one single data segment.
   (B) All the threads within the process share one single heap segment.
   (C) Each thread has its own thread control block.
   (D) Each thread has its own stack space.

5. Which of the following is (are) true about processes and program files? _____

   (A) A process can execute at most one program file during its lifetime.
   (B) A process can execute multiple program files during its lifetime.
   (C) A program file can only be executed in at most one process at a time.
   (D) A program file can be executed in multiple processes simultaneously.

III. Read programs and write down their outputs. We assume each program is run on a computer with a single CPU; when there are multiple processes or threads ready to run, they could run on the CPU in any arbitrary order. So, if there are multiple possibilities in the outputs, provided all; note that, the same information but outputted in different orders are considered as different possibilities.) [10*3=30pts]

(1)

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
int a;
int main()
{
        int c;
        a=1;
        c=2;

        if(fork()==0){
                a=10;
                c=20;
                if(fork()==0){
                        c=30;
                        printf("a=%d, c=%d\n",a,c);
                        exit(0);
                }else{
                        wait(NULL);
                        printf("a=%d, c=%d\n",a,c);
                        exit(0);
                }
        }
        printf("a=%d, c=%d\n",a,c);
}
```

Output:

a=1, c=2
a=10, c=30
a=10, c=20

(2)

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
int a;
int b;
int thread_code(void *arg)
{
	int c;
	b=a+10;
	c=a+20;
}
int main()
{
	int c=3;
	a=1;
	b=2;
	void *p;
	p=(void *)malloc(16384);
	p+=16383;
	clone(thread_code,p,CLONE_VM,0);
	for (int i=0;i<=2;i++){
		printf("a=%d, b=%d, c=%d\n", a, b, c);
	}
}
```

Output:

```
a=1, b=2, c=3
a=1, b=11, c=3
a=1, b=11, c=3
```

(3)

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <stdio.h>
struct shared_data{
    int a;
    int b;
};
int main()
{
    int shmid=shmget(IPC_PRIVATE,
            sizeof(struct shared_data),S_IRUSR|S_IWUSR);
    struct shared_data *p=shmat(shmid,NULL,0);
    p->a=0;
    p->b=0;
    if(fork()==0){
            while(1){
                    if(p->a==0){
                            printf("%d\n",p->b);
                            p->b=1;
                            p->a=1;
                    }
            }
    }else{
            while(1){
                    if(p->a>0){
                            printf("%d\n",p->b);
                            p->b=0;
                            p->a=0;
                    }
            }
    }
}
```

Output:

0
1
0
1

…. (repeated forever)

IV. Write a program that follows such steps. [20pts]
A parent process creates two child processes and one pipe,
child process 1 writes a random number into the pipe,
child process 2 reads the number out of the pipe (and prints it),

In the end, the parent process needs to print out "The child processes have terminated.\n" after detecting the termination of the children.

Note: Don't worry about the #include header statements and the syntax. Also, you can use the library function of rand() to generate a random number.

ANS.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

pid_t parent_id, child1_id, child2_id;
int pipe1[2];

int main() {
   parent_id = getpid();      /* get parent id */
   printf("Parent id is %d\n", parent_id);

   if( pipe(pipe1)<0)    /* generate 1 pipe */
      printf("Error opening pipes\n");

   if( (child1_id=fork()) < 0 )      /* fork 1st child */
      printf("Error creating child1\n");

   if( getpid() == parent_id ) {    /* fork 2nd child */
      if( (child2_id=fork()) < 0 )
         printf("Error creating child2\n");
   }

   if( child1_id==0 ) {        /* in 1st child */
      short ran;
      int buff;
      pid_t cpid1;

      cpid1 = getpid();
      printf("Child1 process id is %d\n",cpid1);

      ran = rand();

       if( write(pipe1[1],&ran,sizeof(ran)) != sizeof(ran) )
         printf("Error writing from child1\n");
```

```
      else printf("Child1 generates random number %d\n and writes
to pipe\n\n", ran);

   }

   if( (child2_id==0) && (child1_id!=0) ) {  /* in 2nd child */
      short number;
      int buff;
      pid_t cpid2;

      cpid2 = getpid();
      printf("Child2 process id is %d\n",cpid2);

      if( (buff=read(pipe1[0],&number,sizeof(number))) <= 0 ) {
         printf("Error reading from child1\n");
      }
      printf("Child2 reads from child1 %d\n", number);



    }

    if( getpid()==parent_id ) {        /* wait for 2 children to
terminate */
      waitpid(child2_id,NULL, 0);
      waitpid(child1_id,NULL, 0);
      printf("The child processes have terminated\n");
   }
}
```

V. CPU Scheduling [20pts]

Consider a SMP computer composed of TWO symmetric processors. These two processors share a common set of process queues. Suppose following processes are submitted to the computer:

Process	Arrival Time	Burst Time (unit)
P1	0	6
P2	1	5
P3	2	3
P4	4	3
P5	5	5

Answer the following questions:

If the round robin scheduling algorithm with time quantum of 2 units is used and hard affinity is required, draw the schedule charts, and answer the following question: what is the average waiting time, the average turn-around time, and the utilizations of the two CPUs? Suppose the CPUs are both idle before these processes arrive, and the context switch time is 0.1 unit.

```
CPU1: P1(0-2)     P3(2.1-4.1)       P1(4.2-6.2)       P3(6.3-7.3)
      P1(7.4-9.4)

CPU2: P2(1-3)     P2(3.1-5.1)       P4(5.2-7.2)       P5(7.3-9.3)
      P2(9.4-10.4)      P4(10.5-11.5)           P5(11.6-13.6)
      P5(13.7-14.7)
```

Ready Queue:

Time	Process	
0	P1	P1----C1
1	P2	P2----C2
2	P3,P1	P3----C1
3	P1,P2	P2----C2
4	P1,P4,P3	P1----C1
5	P4,P3,P5,P2	P4----C2
6	P3,P5,P2,P1	P3----C1
7	P5,P2,P1,P4(7.2) (P3 FINISHED AT 7.3)	P5----C2 P1----C1
8	P2,P4	
9	P2,P4,P5 (P1---FINISHED)	P2-----C2
10	P4,P5 (P2--- FINISHED)	P4-----C2

11	P5 (P4----FINISHED)	P5-----C2
12		
13	P5	P5-----C2
14	P5-----FINISHED	

Average Waiting Time:

[(9.4 – 6 - 0)+(10.4 – 5 - 1)+(7.3 – 3 - 2)+(11.5 – 3 - 4)+(14.7 –5-5))]/5
= 19.3/5 = 3.86

Average Turnaround Time:

[(9.4-0)+(10.4-1)+(7.3-2)+(11.5-4)+(14.7-5))]/5
= 41.3/5 = 8.26

Utilizations:

CPU1: 9/9.4 = 95.75%

CPU2: 13/14.7 = 88.43%

Your score is: _____