Department of Electrical Engineering, IIT Delhi
EDL358 Operating Systems: Minor I Examination
(Closed Book/Closed Notes) Time: 1 hour Maximum Marks: 25

"Thou shalt not covet thy neighbour's answers"

1. **Brevity is the Sole Soul of Wit: Short Answers only, please!**

   (a) After a PC is switched on, what happens initially when the Program Counter/Instruction Pointer of the microprocessor is asynchronously set to zero, and when and how does an OS actually kick in?

   (b) Suppose an input device is sending data to a computer, and a time-sharing OS decides to take away control of the CPU away from the process that was taking the input. What will happen?

   (c) What is the fork bomb problem, and what happens on a Linux machine when exposed to a fork bomb?

   (d) A fundamental difference in Windows and Linux schedulignis that Linux gives high priority processes largr time quanta, while Windows does the opposite. State the advantage of each. *(3+2+3+2 marks)*

2. **A `fork()` problem: Food for Thought** Consider the following code

```
int main(void)
{
pid_t pid; int ret_val;
unsigned char value; unsigned char * address;

value = 'a';
address = (unsigned char *) malloc(sizeof(unsigned char));
*address = 'a';
printf("PARENT: value=%c (address %d),address %d has=%c\n",
        value, (int) &value, (int) address, *address);
switch(pid = fork())
{ /* --- switch case --- */
case -1: /* --- error --- */
        perror("fork"); exit(1);
case 0: /* --- child process --- */
        printf("CHILD: pid=%d\n", getpid());
        printf("CHILD: parent's pid=%d\n", getppid());
        printf("CHILD: value=%c (address %d), address %d has=%c\n",
                value, (int) &value, (int) address, *address);
        value = 'b'; *address = 'b';
        printf("CHILD: value=%c (address %d), address %d has=%c\n",
                value, (int) &value, (int) address, *address);
        exit(1);
default: /* --- parent process --- */
        printf("PARENT: pid=%d\n", getpid());
        printf("PARENT: parent's pid=%d\n", getppid());
        printf("PARENT: child's return status=%d\n",
                WEXITSTATUS(ret_val));
        printf("PARENT: value=%c (address %d), address %d has=%c\n",
                value, (int) &value, (int) address, *address);
} /* --- switch case --- */
free(address);
return 0;
}
```

value = 'a'

printf("PARENT: child's pid = %d\n", pid);

A sample run of the above code segment gives the following actual output, with exception of the 2 characters each in lines 7, 8 and 11, where they have been replaced with a '@':

```
PARENT: value=a (address -262003441),address 7598096 has=a
PARENT: pid=2554
PARENT: parent's pid=2214
PARENT: child's pid=2555
CHILD: pid=2555
CHILD: parent's pid=2554
CHILD: gets value=@ (address -262003441), address 7598096 has=@
CHILD: sets value=@ (address -262003441), address 7598096 has=@
Please enter ret_val:1
PARENT: child's return status=1
PARENT: value=@ (address -262003441), address 7598096 has=@
```

(a) Why are some address values negative? Explain. *(2 mark)*

(b) Why are addresses the same in the parent and child processes? *(2 marks)*

(c) What will be the actual output, in place of '@' on line 7? *(2 mark)*

(d) What will be the actual output, in place of '@' on line 8? *(2 mark)*

(e) What will be the actual output, in place of '@' on line 11? *(2 mark)*

(f) How many processes result in the following segment of code? Explain

```
int main(void){ fork(); fork(); fork();}
```

(2 marks)

3. **A stitch in time...threads**

```
#define NUM_THREADS     5
int var1 = 5;
void *TaskCode(void *argument)
{
   int tid;
   tid = *((int *) argument);
   printf("Hello World! It's me, thread %d!\n", tid);
   return NULL;
}
int main(void)
{
   pthread_t threads[NUM_THREADS];
   int thread_args[NUM_THREADS];
   int rc, i, var2 = 5;

   for (i=0; i<NUM_THREADS; ++i) { thread_args[i] = i;
   printf("In main: creating thread %d\n", i);
   rc = pthread_create(&threads[i], NULL, TaskCode, (void *) &thread_args[i]); }
   for (i=0; i<NUM_THREADS; ++i) { rc = pthread_join(threads[i], NULL); }
   exit(EXIT_SUCCESS);
}
```

Just before the `return NULL;` line of the thread TaskCode, suppose a programmer inserts the following line. What happens? *(3 marks)*

```
printf("var1 = %d, var2=%d\n",var1,var2);
```