Department of Electrical Engineering, IIT Delhi

ELL405 Operating Systems: Minor I Examination

(Closed book/Closed Notes) Time: 1 hour Maximum Marks: 25

"Thou shalt not covet thy neighbour's answers"

1. **Processes and threads!** Consider the following program:

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
int sum=0;
void * runner (void * param);
int main(int argc, char *argv[])
{
pthread_t tid;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_create(&tid,&attr,runner,argv[1]);
fork();
pthread_join(tid,NULL);
printf("sum = %d\n",sum);
}
void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;
printf("inside thread\n");
for (i = 1; i <= upper; i++) sum+= i;
pthread_exit(0);
}
```

What should running the compiled code on a Linux system `a.out 3` produce, as output? Explain *((2+1) marks)*

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

(a) In designing an instruction set for a computer, what type of instruction is absolutely essential, for an OS to exist?

(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) Differentiate between object code, executable code and library code

(d) Differentiate between static linking and dynamic linking. What happens when two or more processes link to a dynamically linked library?

(e) Suppose the main thread does not wish to wait for the completion of its child thread, using `pthread_join()`. Give an example of how the thread can exist after its parent thread has exited.

(f) The reason for thread-based scheduling in an OS is that if a thread in a process blocks, in process-based scheduling, the entire process blocks. Suppose an OS can detect a blocking thread, it can automatically switch to the next. How will this impact the average turnaround times in process-based scheduling, and thread-based scheduling, respectively? *(2+2+2+(2+1)+2+2 marks)*

3. **Memory matters!**

   (a) First consider a non-shared memory case. A programmer creates a pointer to an integer, allocates memory to it, uses it, and then frees it. However, the programmer uses it again, now. Explain the possibilities that could happen.

   (b) Now, the programmer sets the pointer to (int *)NULL just after freeing it. What happens if the pointer is used again?

   (c) Suppose a program has a memory allocation, which is not freed before the program terminates. What happens to that allocated memory?

   (d) **Shares pay dividends!** Consider the following program:

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main(void)
{
int segment_id;
char * shared_memory;
const int size = 4096;
segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);
printf("segment ID:%d\n",segment_id);
shared_memory = (char *) shmat(segment_id, NULL, 0);
sprintf(shared_memory, "Hi there!");
while(1) { }
return 0;
}
```

   If the same segment_id is entered in the input of another program (below), will it work? Please explain. *(3+1+2+3 marks)*

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main(void)
{
int segment_id;
char * shared_memory;
const int size = 4096;
printf("Please input segment ID:\n");
scanf("%d",&segment_id);
shared_memory = (char *) shmat(segment_id, NULL, 0);
printf("%s\n",shared_memory);
return 0;
}
```