M.Sc. (Informatics) 2nd Semester, 2014
Paper: IT-23 (Operating Systems)

Time allowed: **03 Hrs** Maximum Marks: **75**

(Answer Question Number 1 and any 4 from the rest)

Q1)

a) Considering the following states of a process, draw a state transition diagram. Also clearly mention the *events* on which state transitions take place.

Process States – *new, ready, running, waiting, terminated.*

b) Briefly mention the similarities and differences bwteen a "PIPE" and a "NAMED PIPE"

→ FIFO

c) Consider the following code fragment,

```
#include<stdio.h>
#include<unistd.h>
int main()
{
        printf("Hello\n");
        fork();
        fork();
        fork();
        fork();
        printf("Hello\n");
        return 0;
}
```

PIPE: cannot be used by processes that do not have common ancestry (e.g. parent child); unidirectional; opened when created; two fd's (file descriptors fd[0] for read & fd[1] for write) returned by open fn.

FIFO: named pipe can be used b/w two totally unrelated process to communicate; bidirectional; first created then opened for writing & reading; one fd (returned by open() fn).

How many times the string ""Hello\n" will be printed and why? 8 times.

d) Briefly mention the differences between a "static" and "dynamic" library? What are the commands you can use to create static and dynamic libraries in a Linux distribution?

e) In the context of IPC (Inter Process Communication) objects, what is the difference between

"***Process Persistence***" and "***Kernel Persistence***"?

mechanism

last until all the processes that opened it closes it, exit or crash, eg. pipe

exists until the kernel of OS reboots or is explicitly deleted eg. semaphore

[3 * 5]

Q2)

a) Consider the following code fragment and then answer the questions.

Process persistent → PIPE, FIFO
Kernel persistent → message queue, semaphore, shared memory.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
char *c ;
int main()
{
        int ret = -1;
        ret = fork();
        if( ret == 0)
```

ar rcs lib___.a ← .o → static linking

gcc -shared -o lib___.so ← .o → dynamic linking

```
{
        printf("Child : \n");
        c = malloc(sizeof(char));
        *c = 'A';
        printf("The value in variable c is %d\n", *c );
        exit(0);
}
else
{
        printf("Parent :\n");
        printf("The value in variable c is %c\n", *c + 1);
        free(c);
        exit(0);
}
}
```

./aname executing in subshell & after execution, reflects the change in sub-shell but not in main shell. '. aname' or 'source ./aname' then in main shell and reflects the changes.

i) What is the purpose of the "fork()" system call? What are the return values of this system call? Mention the return value if this call fails, for some reason. [3]

PID of child returned to parent, 0 to child else -1 returned to parent, no child created.

ii) What do you think will be the output of the program? Is there any chance of run-time memory fault, if this source code is compiled and executed? Explain briefly. [4]

run-time memory fault in Parent (∵ c not allocated in Parent)

iii) The library function "malloc" allocates memory dynamically. What will happen to the allocated memory when a process terminates? (Suppose library function "free" is not used explicitly to free memory.) [2]

Memory leak

b) Suppose there is an *external command* "mycommand" available in a standard "Linux" like operating system distribution. What do you mean by external command? What may be the other type (s) of command? Support your answer with an example. How can you distinguish between different types of commands? (Mention any command to do this) [6]

→ ls, wc → new subprocess formed. (fork() and execlp → a child process)

Internal commands → echo, pwd, cd → no separate process called, built in shell interpreter

⇒ type cd — cd is shell builtin

⇒ type wc — wc is /usr/bin/wc → path from where wc is loaded.

* The path to find the executable of ext commands must be provided in OS kernel variable called Path variable — multiple directories by ':' — 'type' command used to find external/internal

Q3)

The file listing command "ls -li" is being executed in a "Linux" Shell and the output is shown below. Study the output and then answer the questions.

[localhost msc_iic]$ ls -li

656158 -rw-rw-r-- 3 user user 0 Feb 9 2013 file4

656158 -rw-rw-r-- 3 user user 0 Feb 9 2013 file2

656158 -rw-rw-r-- 3 user user 0 Feb 9 2013 file1

656160 lrwxrwxrwx. 1 user user 5 Feb 9 2013 file3 -> file1

3rd column → no. of links with that file. eg. file4, 2, 1 linked ∴ 3 links.

regular file → link = 1
directory → links = 2 [.. parent directory, . (dot) entry signifying the current directory referring the same inode]

---

i) What is the purpose of the "-li" after "ls"? [2]

l → Long listing format, i → inode no. (Index node no.)

ii) What is the significance of the first column of the output? For the first three rows the value of the first column is the same. Explain this observation. [5]

iii) What is the significance of the 3rd column of the output? Briefly mention how this will be different when you create a "regular file" and when you create a "directory". For the first three rows the value is 3 and for the fourth row the value is 1. Explain this observation. [8]

→ Inode no.: Inode is a data structure on linux file system & inode no. is an index to a very large array of inode structure. Every element of that array store the file's attributes (like type of file, no. of links etc.) inode value same ∴ both hardlinks with each other, point to same inode str & data. If 1 modified, then changes in 2nd

Q4)

a) Briefly mention the purpose of a "make file". Write a makefile from the description given below.

"The name of your final executable should be myoutput. It depends on four object files main.o, f1.o, f2.o, f3.o. These object files depend of source files named as main.c, f1.c, f2.c, f3.c. There is a single header file "myheader.h" which is referred by all these source files. The compiler that you need to use is "gcc". Also you need to specify a target which can remove all the object files and the final executable."

[5]

b) Consider the following code fragment (ignore the absense of any header files), which uses a System V Shared Memory. Answer the question that follows. [10]

```
int main ( )
{
        int shmid;
        key_t key;
        char *virtualaddr;
        pid_t ret;
        key = ftok("shared_simple.c",'R');
        shmid = shmget(key,1024,0644|IPC_CREAT);
        virtualaddr = shmat(shmid,(void*)0,0);
        strcpy(virtualaddr, "parent");
        ret = fork();
        if (0 == ret)
        {
                strcpy(virtualaddr, "child");
                exit(0);
        }
        wait(NULL);
        printf("%s", virtualaddr);
        exit(0);
}
```

i) What will be the output of this code? Parent.

ii) If the "wait" call is not present will the output differ? → yes. there will be a zombie process. (parent exits w/o child)

iii) Briefly mention what is the purpose of the ***ftok, shmget*** and ***shmat*** calls.

↳ generates a key → from path name and process ID. If one process needs to share the memory. It gives same path name & process ID and gets the same key.

Q5)

a) Consider the following code fragment (ignore the absense of any header files), which tries to invoke the standard word count program (wc) with the help of the "execlp" system call. Answer the question that follows.

```
int main()
{
    char *beforeexec = "Before Calling Exec";
    char *afterexec = "After Exec";
    write(1, beforeexec, strlen(beforeexec));
```

ftok() is used ∴ If we give a random no. to key and if one unrelated process also has the same no., it can share the area (which we donot want). ∴ ftok() used.

```
    execlp("wc","wc","call_wc.c",NULL);
    write(1, afterexec, strlen(afterexec));
    return 0;
}
```

i) Briefly state the working of the "execlp" system call. What is the meaning of "l" and "p" in "execlp"? Why is "wc" written twice in the arguments of execlp? What is the return value of "execlp" call? [3 + 2 + 2 + 2]

b) Consider a semaphore variable S. Define the operations **Wait(S)** and **Signal(S).**

Suppose two processess P1 and P2 are two be scheduled in such a way that statement S2 in Process P2 will be executed only after statement S1 of Process P1. Implement this requirement with the help of a semaphore. [4 + 2]

Q6)

a) What is significance of the Shell (assume a standard Shell like bash) Environmental Variable PATH? How can you print the value of PATH variable? [2]

b) Consider the following code fragment (ignore the absense of any header files), which tries to read data from a text file named "example.txt". Assume that the file is created using an editor, having five characters written "abcde". The code compiles successfully. Answer the questions that follow. [8]

```
#define MAXBUF 10
int main()
{
    int fd;
    int ret;
    char buffer[MAXBUF];
    fd = open("example.txt", O_RDONLY);
    printf("%d\n", fd);
    ret = read(fd, buffer, 5);
    printf("%d\n", ret);
    printf("%s", buffer);
    exit(0);
}
```

i) What do you mean by "file descriptor" ?

ii) What will be value printed for "fd" and why?

iii) What will be the value printed for "ret" and why?

iv) It is seen that while printing the "buffer", the output is not proper. Why? How can you correct this?

c) Briefly mention the features of a System V Message Queue. "Multiple processes can access the same queue." How can you achieve this?

Mention a command which can be used to delete a message queue. (Assume you are using a standard linux distribution) [5]