Birla Institute of Technology & Science, Pilani Hyderabad Campus CS F372: Operating Systems (Assignment 1)

Assigned: 29.01.2026 Date of Submission: 11.02.2026 Total Marks: 12 Weightage: 6%

General Instructions:

• This assignment can be done in groups of **maximum 4** students.

• All your programs must run on a Ubuntu 24.X virtual machine on Oracle Virtual Box. • Recommended allocation: 4 cores, 50 GB virtual disk, 4GB RAM or more on VM. • Submit all source files and executables along with a readme that has group details. • Put all your deliverables in a tar file like f2023xxxx.tar and submit through course webpage (assignment secion)

• Every assignment will have a demo and viva, the schedule of which will be intimated later through a separate notice.

 For any assignment related queries, you may contact I/C or the following teaching assistants:

• Arpit Kudre <f20230158@hyderabad.bits-pilani.ac.in>
• Parth Munjal Joshi <f20230127@hyderabad.bits-pilani.ac.in>

Q1. (2 Marks)

Write a C program that forks a child process:

Input and Array
  • The parent process takes an input **'n'** from the user. Assume **n** is more than 0. • The parent also has an integer array containing some integers, say {3,15,4,6,7,17,9,2}.
Loop Behaviour
  • The parent process selects any one number randomly (using **rand()** or **srand()**), say x, from the array, prints it and passes it to the child process via an unnamed pipe. • Then it proceeds to sleep for **x%n** seconds.
  • The child computes and prints out all the factors of the number (including 1 and the number itself).
  • The child sleeps for **time(NULL)%n** seconds (i.e. **UNIX_time % n**).
Process Termination
  • The process should stop once every integer is visited at least once, or • after an outside interrupt (like **Ctrl+C**) is received.

Do not use any global variables.

Q2. (3 Marks)

A memory monitoring system is a utility program that continuously observes system resource consumption and provides mechanisms to manage processes based on their memory usage, serving as a practical tool for system administrators to prevent resource exhaustion. This dual requirement, automated surveillance combined with human decision-making, is elegantly addressed through a parent-child process architecture.

Write a C program mimicking a memory monitor:

The program should fork a child process which would be responsible for monitoring the memory usage and killing selected processes while the parent would handle user interaction.

Input Variables

The program takes 3 integers n, k, r as input from the user. Assume n,k,r to be more than 0.
- **n**: time in seconds between 2 consecutive prints
- **k**: processes displayed in each print
- **r**: iterations after which a process is killed

Loop Behaviour
- The child process prints **k processes** with the highest memory usage every **n seconds**.
- The output must be printed in descending order of memory usage in the following format.

```
USER          PID %MEM     TIME
```

- You may use pipe, fork, exec family, dup2 and sleep system calls to implement the program as needed.

Parent-Child Coordination
- After **r iterations** (i.e., after the child has printed the top k processes r times), · Child process halts printing and waits for input through pipe of parent · The parent prompts the user to enter the PID of the process to kill. · It sends this input to the child using a pipe.
- The child receives the PID and does either of the following:
  - If the input is -2 it exits the program
  - If the input is -1 skips killing
  - Kills the process with PID equal to the input otherwise
- After killing, the child process resumes printing the memory usage list.

Program Termination
- The program only terminates if user inputs -2 or,
- after an outside interrupt (like **Ctrl+C**) is received.

Do not use any global variables. Use ps with BSD syntax to obtain memory usage and format output appropriately.

Q3. (2 Marks)

Grep Line Number Extractor:

Grep is an acronym for *global regular expression print*, It is a powerful command-line utility in Unix-based systems used to search for lines that match a specific pattern (regular expression) within files or input streams and print the matching lines to standard output.

Example Command - Running grep on a single file prog.c:
bash$ grep -n int prog.c

Sample Output:
12: int a = 1;
17: printf("%d", a);
34: char s[10] = "integer";

Where, each matching line is preceded by a line number and a colon. The behaviour is slightly different when grep is run on multiple files.

Example Command - Running grep on multiple text files:
bash$ grep -n Hello belgium.txt france.txt

Sample Output:
belgium.txt:1:Hello, This is Belgian
belgium.txt:3:Hello with Waffles
france.txt:1:Hello, This is French
france.txt:2:I am saying Hello in Croissants

Now write a program mygrep which prints just the line numbers sorted in ascending order with duplicate line numbers removed.

Example Command – Running mygrep (assuming same files in cwd as above example): **bash$ mygrep Hello belgium.txt france.txt**

Sample Output:
1
2
3

[NOTE: You are NOT supposed to implement the regular expressions, duplicates removal, sorting logic in your C code. You are instead expected to use UNIX commands such as grep, cut and sort and get this desired output through constructs you learnt in this course such as forks, pipes, exec .etc. and let your C code route their outputs]

You can assume that the maximum size of input to mygrep will be 1024 bytes.

Q4. (3 Marks)

Developing a shell :

The shell or command line interpreter is the fundamental user interface to an Operating System. Every shell is structured as the following loop:

1) print out a prompt
2) read a line of input from the user
3) parse the line into the program name, and an array of parameters
4) use the fork() system call to spawn a new child process
   a) the child process then uses the exec() system call to launch the specified program
   b) the parent process (the shell) uses the wait() system call to wait for the child to terminate
5) When the child (i.e. the launched program) finishes, the shell repeats the loop by jumping to step 1.

Write a simple shell in C used to interact with a cellular base station controller named bs_shell that has the following properties:

Internal Commands: These are commands directly executed by the shell process. (no need to fork) (in linux, type <command> gives information about internal or external.)

· **set_freq <MHz> -** Configure the base station carrier frequency to <MHz>. Store this val-ue in a local variable. If the frequency is non-positive integer show an error msg. · **get_freq -** Display the current base station carrier frequency.
· **quit -** allows us to exit the shell

External Commands: These are commands to execute which the shell has to fork, the child process sends an exec() call while the parent waits for the child to complete execution. (as detailed in the earlier loop given in the question)

· **top -** displays information about running processes and system resource usage at time of execution.
· **ping <address> -** tries to ping given address exactly 4 times. (Eg address: google.com).

Additional Requirements:

· The shell prompt should be: **station-controller$**
· Initialize frequency to 800 MHz
· Show error message for commands not defined above.
· You should handle SIGINT signal (Ctrl+C). During normal operation, if user presses Ctrl+C, the shell should not terminate. Instead, it should print*: [WARNING] Carrier Interrupt Signal Received,* reset frequency to default safe mode(800 MHz)*,* continue normal operation and only quit command can exit the shell.
· Assume maximum input in a line is 1023 characters.

[Hint : To split the command and the arguments you can look into strtok function man page.]

Q5. (2 Marks)

A system call is a programmatic interface that allows applications to request services from the operating system's kernel, acting as a controlled gateway between user programs and privileged system resources as discussed in the class. When a program needs to perform operations it cannot do directly-like reading files, creating processes, or accessing hardware it makes a system call that triggers the CPU to switch from user mode to kernel mode, allowing the OS to safely execute the requested operation. This mechanism exists to maintain system security and stability by preventing direct access to critical resources while still enabling programs to perform essential tasks like file management, process control, device access, and inter-process communication. After the kernel completes the operation, control returns to the user program along with any results.

The procedure to create a system call varies in different versions of Linux. When you add a system call you will have to recompile the Linux kernel, so you must have the Linux source tree installed on your system.

Adding a System Call to the Linux Kernel:

Add a system call to the Linux kernel *(version: 6.14)*, called **myfork**, whose functionality is same as the fork () system call, except it also prints the process-id of the parent process that called the fork to the kernel buffer.

Use the following function to test your system call.

```
void myfork_test () {
long result = myfork ();
printf ("Hello World!");  return; }
```

Here myfork () is a local wrapper to a syscall you define in the kernel before compiling it.

```
#include <sys/syscall.h> long myfork(void) {



return syscall (467); //assuming you added myfork as syscall 467
}
```

To test if message was in kernel buffer run the command: **dmesg | grep**

myfork Helpful Resources:

• [linux-source-tree]
• [fork new method – linux kernel 6.14]
• [fork exact implementation](Line 2681-2694)
• [printing to kernel buffer – linux kernel 6.14]
• [obj-y in make files]
• [syscall tables and headers]

(Computer Sc. Notice Board) (C. R. Hota, I/C) **-x**