CS333:Operating Systems Lab
Autumn 2023
LabQuiz 1, 23 marks

- **No Internet. No phone. No devices. No handwritten notes. No books.**
- **No friends** (strictly for the duration of the exam only!)
- **Please read the submission instructions carefully.**

- **Solve any one of Question 2 or Question 3.**

Question 1	Question 2 or Question 3	Question 4
5 marks	8 marks	10 marks

Q1. divide and conquer with fork (5 marks)

p1.c is a program that reads N integers from an input file and stores them in an array A. Check input files provided for format.

Extend p1.c (after setup of the array) to spawn 2 child processes— the first child process prints the sum of the elements present at **odd** indexes in the array along with its PID and PPID, while the second child process prints the sum of the elements present at **even** indexes in the array along with its PID and PPID. **[3 marks]**

The parent process exits after reaping both the child processes, and prints its own pid and the pid of reaped children. **[2 marks]**

Notes

- Format for printing by child process : `"pid=12345, ppid=12344, sum=9"`
- Format for printing by parent process: `"pid=12344, pid_reaped=12345"`
- Output should format should be as mentioned
- Output from child process can be in any order

Sample test case 1

Input

```
6
1 2 3 4 5 6
```

Output

```
pid=12345, ppid=12344, sum=9
pid=12346, ppid=12344, sum=12
pid=12344, pid_reaped=12345
pid=12344, pid_reaped=12346
Number of children not reaped yet: 0
```

The first child process prints the sum of all the odd indexed elements (1 + 3 + 5 = 9), while the second prints the sum of all the even indexed elements (2 + 4 + 6 = 12). All child processes are reaped before exit of parent process, and the last line is printed as shown.

Sample test case 2

Input

```
5
3 5 4 5 2
```

Output

```
pid=21345, ppid=21344, sum=9
pid=21346, ppid=21344, sum=10
pid=21344, pid_reaped=12345
pid=21344, pid_reaped=12346
Number of children not reaped yet: 0
```

The first child process prints the sum of all the odd indexed elements (3 + 4 + 2 = 9), while the second prints the sum of all the even indexed elements (5 + 5 = 10). Since all child elements are reaped, the last line is printed with the number zero.

To submit: `p1.c`

Q2. hub-and-spoke with pipes (8 marks)

p2.c is a program that reads N integers from an input file and stores them in an array. Check input files provided for format.

Extend **p2.c** (after setup of the array) to spawn **N** child processes. Each child process should double the value stored in i^{th} index of array **A** and send it back to the parent process through a pipe (You will need **N** pipes to run this in a distributed manner). **[3 marks]**

Each child process should print its pid, ppid and the value of the doubled element. **[3 marks]**

Parent process should reap all the children and prints its pid and the sum of the modified array.**[2 marks]**

Notes

- Format for printing by child process : `"pid=12345, ppid=12344, element=9"`
- Format for printing by parent process: `"pid=12344, sum=100"`
- Output should be in the exact format as mentioned.
- Output from child process can be in any order

Sample test case 1

Input

```
5
1 2 3 4 5
```

Output

```
pid=12345, ppid=12344, element=2
pid=12346, ppid=12344, element=4
pid=12347, ppid=12344, element=6
pid=12348, ppid=12344, element=8
pid=12349, ppid=12344, element=10
pid=12344, sum=30
Number of children not reaped yet: 0
```

Parent process creates 5 child processes. Each child process prints its pid, ppid and twice the value stored in the i^{th} index of array A. Finally the parent process prints its pid, and the total sum = 30. Since all child elements are reaped, the last line is printed.

Sample test case 2

Input

```
1000
2 4 6 8 10 12 14 16 18 20 . . .
```

Output

```
pid=21345, ppid=12344, element=4
pid=21346, ppid=12344, element=8
pid=21347, ppid=12344, element=12
pid=21348, ppid=12344, element=16
pid=21349, ppid=12344, element=20
.
.
.
pid=21344, sum=2002000
Number of children not reaped yet: 0
```

Parent process creates 1000 child processes. Each child process prints its **pid**, **ppid** and twice the value stored in the i^{th} index of array A. Finally, the parent process prints its **pid**, and the total sum = 2*1000*1001 = 2002000. Each child sends its vaSince all child elements are reaped, the last line is printed.

To submit: `p2.c`

Q3. backing background execution (8 marks)

Implement a simple shell program to support the execution of multiple commands in the background, as described below.

- Multiple commands separated by &&& should be executed in parallel in the background. That is, the shell should start execution of all commands simultaneously, and return to the command prompt to accept new commands. **[3 marks]**
- Handle appropriate reaping of background child processes.
  Hints:
  - Which signal is delivered to the parent process when a child process terminates? **[2 marks]**
  - If multiple children end at the same time/in quick time, and the parent process has not processed the signal yet, the signal is set only once for the parent. How to handle the signal such that all zombie children are reaped? **[3 marks]**
  - **wait (by extension waitpid)** by itself is a blocking call, i.e., it suspends the process until one of its children terminates (given that child exists). On the other hand, waitpid can be made non-blocking with the WNOHANG option. e.g., `waitpid(1234, NULL, WNOHANG)` would return immediately if the child process with pid 1234 hasn't terminated.
- Some commands/programs expected to work for individual and parallel execution are: `ls, pwd, ps, clear, sleep`. (cd will not be used for this question)
- Pressing CTRL+C should exit the shell.
- Individual commands run in the foreground

You are provided with a boilerplate code **(p3.c)** which contains a tokenizer and hints to guide you through the implementation. Note that you are required to add only this functionality as part of the shell and not other shell related features. Assume that there are spaces on either side of the special token &&& and no more than 64 commands will be given at a time.

```
kanishka@LAPTOP-KNATN80T:/mnt/c/Users/kanishk
$ ls
file.txt  p1  p1.c  p3  p3.c  p4
$ sleep 5 &&& sleep 10
$ ps -a
  PID TTY          TIME CMD
16414 pts/1    00:00:00 p3
16416 pts/1    00:00:00 sleep
16417 pts/1    00:00:00 sleep
16418 pts/1    00:00:00 ps
$ ps -a
  PID TTY          TIME CMD
16414 pts/1    00:00:00 p3
16416 pts/1    00:00:00 sleep
16417 pts/1    00:00:00 sleep
16419 pts/1    00:00:00 ps
$ ps -a
  PID TTY          TIME CMD
16414 pts/1    00:00:00 p3
16417 pts/1    00:00:00 sleep
16420 pts/1    00:00:00 ps
$ ps -a
  PID TTY          TIME CMD
16414 pts/1    00:00:00 p3
16421 pts/1    00:00:00 ps
$
```

You may want to test with multiple long running commands (e.g., via **sleep)** to test your implementation, as such commands will give you enough time to run **ps** in another window and in the same shell to check that the commands are executing as specified.

Sample test case and output

ls runs in the foreground and gives the output

sleep 5 &&& sleep 10 starts 2 background processes in the background
ps -a confirms 2 background child processes
ps -a after 5 seconds shows only 1 sleep process because the other child has been reaped
ps -a after 10 seconds shows no sleep processes

To submit: `p3.c`

Q4. process scheduling with signals (10 marks)

Signals can be used to control the execution of processes (stop, continue, terminate etc.). The aim of this exercise/question is to build a simple scheduler—with the help of signals—to alternate between the execution of two processes. The input parameters of the scheduler program are as follows:

- `time-slice`: The duration (in seconds) for which the scheduler should allow each process to execute before switching to the other process.
- `c1-execution-time`: Total duration in seconds for which child 1 would execute.
- `c2-execution-time`: Total duration in seconds for which child 2 would execute.

Modify the `p4.c` file provided in `auxiliary files` for the following requirements,

- The program should first spawn two child processes and immediately put them in the WAITING state. Each child executes the `child_process` function (provided in the boilerplate code) with the respective execution time provided as input parameters. **[3 marks]**
- Next, the scheduling logic should take over and schedule one child process at a time allowing them to run for `time-slice` duration in each chance. **[3 marks]**
- If one of the child processes terminates while the other is still active, the scheduler should allow the second process to execute till completion (as no slots are required for the terminated child). The scheduler should make use of signals for controlling the state of processes. **[3 marks]**
- The scheduler program should terminate only after both the child processes are terminated. **[1 mark]**

Hint: wait (by extension waitpid) by itself is a blocking call, i.e., it suspends the process until one of its children terminates (given that child exists). On the other hand, waitpid can be made non-blocking with the WNOHANG option.

Syntax:

```
./p4 <time-slice> <c1-execution-time> <c2-execution-time>
```

Sample Test Case:

```
./p4 2 6 10
timeslice: 2, c1 execution time: 6, c2 execution time: 10
Child [17083]: 1
Child [17083]: 2
Child [17084]: 1
Child [17084]: 2
Child [17083]: 3
Child [17083]: 4
Child [17084]: 3
Child [17084]: 4
```

```
Child [17083]: 5
Child [17083]: 6
Child [17084]: 5
Child [17084]: 6
Child [17084]: 7
Child [17084]: 8
Child [17084]: 9
Child [17084]: 10
Parent [17082] terminates!!
```

To submit: `p4.c`

Submission instructions

- Auxiliary files are available in the `auxiliary_files` folder.
- For submission, create a folder on the Desktop named: `submission_<rollno>`
  E.g., a directory named `submission_22d0371`

  Please strictly adhere to this format otherwise your submission won't count.

- Put all the submission files (mentioned above in each question under **To Submit**) in the aforementioned folder **(only one of p2.c or p3.c needs to be present).** Your folder structure should look something like this:

```
submission_<rollno>/
            ├──── p1.c
            ├──── p2.c
            ├──── p3.c
            └──── p4.c
```

- Next, open the terminal and run `check` command. This should ensure that the submission folder exists and is not empty. Further, this generates the required tar file and places it on the Desktop. **(This step for student and TA)**
  Note: The check program is pre-installed and is not part of the auxiliary files

- Make sure that the required tar file is visible on the Desktop, and then run the `submit` command. You'll require a password for the final submission which the TAs will use. **(This step is for TAs only).**

- **Due Date: 24th August 2023, 4:55 pm**