CSCI4730/67360 – Operating Systems

PA #1: Multi-process and IPC

Due date: 11:59 pm, Feb/13/2026 (Friday)

Description

In this programming assignment, your objective is to design and implement a multi-process version of the "word-count" program. While the single-process version has been provided, your task involves transforming it into a multi-process architecture."

Part A: Multi-process "Word-Count" Program

The main limitation of a single-process program lies in its scalability, as it struggles to efficiently process a large volume of words.

To overcome this limitation, you will convert the "word-count" program into a multi-process model. In this new "multi-process" model, the main process will create multiple child processes, effectively distributing the workload among them. These child processes will communicate their individual results back to the main process through an Inter-process communication (IPC) channel. We will use **pipe** in this assignment. As the main process waits for the child processes to complete their tasks, it will read the results via the IPC channel and subsequently display the aggregated total on the screen. Here are your action items:

- You will modify "**wc.c**" to a multi-process model.
- I recommend you to use "**wc_mul.c**" template.
- The program receives the number of child processes and an input file name through the command-line argument.
  - 1st argument: # of child processes (default is 0)
  - 2nd argument: target file
  - 3rd argument: crash rate (default is 0%)
  - Example: **./wc_mul 4 large.txt 20**
- Explain your program structure and IPC in README.pdf file. **Only "pdf" format** will be accepted.
- **To get full credit for part A, the multi-processing version of word-count should provide a significant performance improvement.**

Part B: Crash-handling

In the event that one or more child processes terminate unexpectedly before they can transmit their results through **pipe**, the final output may become inaccurate. In this assignment, we will monitor the exit status of each child. If there is one or more child processes crashed, we will create new child processes to complete the job.

The parent process should "wait" for all child processes and monitor their exit status (hint: use **waitpid()**). If an exit status of a child process is "abnormal termination" by a signal, the parent process creates a new child process to re-do the incomplete job.

- You can use 3rd command-line arguments (integer between 1 and 50) to trigger crash. 50 means each child process has 50% chance to be killed abnormally by signal. 0 means 0% chance to crash.
- Explain how your program handles crash in README.pdf file.

Grading

	Undergrads	**Grads Students**
Part A	**70**	30
Part B	30	**70**
Total	100	100

Submission

Submit a tarball file using the following command

%**tar czvf p1.tar.gz README.pdf Makefile wc_mul.c**

1. README.pdf file with:
    a. Your name
    b. Explain your design of multi-process structure and IPC.
    c. Explain how your program handles crash.
2. Your code should be compiled and tested on **odin** server (odin.cs.uga.edu).
3. Submit a tarball through ELC.
4. The late submission policy is applied.