Hacettepe University Computer Engineering Department
BBM342 Operating Systems
Experiment 2

Subject:	Multi-Thread Programming, Inter Process Communication, File System Calls
Advisors:	Assoc. Prof. Dr. Ahmet Burak Can, Assoc. Prof. Dr. Harun Artuner, Asst. Prof. Dr. Kayhan İmre, Dr. Ali Seydi Keçeli, Dr. Aydın Kaya
Language:	ANSI C
Submission Date:	04.04.2017
Deadline:	28.04.2017

---

1. Introduction

The aim of this experiment is to make you familiar with processes and threads, and have practical experience with multi-process and multi-threaded programming using UNIX and POSIX functions.

2. Experiment

In this experiment, you will develop a parallel File Search system. You have to use POSIX P-Threads, mutexes, and Inter-Process Communication in UNIX (Pipes).

A general schema for the system is given in Figure 1. File searcher's main process and the minion processes are represented with distinct processes. The File Searcher Main Process will be multi-threaded where each Controller Thread communicates with Minion Processes through pipes. The File Searcher Main Process will manage the File Buffer that is filled by File Searcher Thread and emptied by Controller Threads. The File Buffer will be fixed size, which will be provided as a command line argument to the File Searcher Main Process. Accessing the File Buffer requires synchronization of threads.

The File Searcher Main Process starts with four command line arguments. First one is number of the Minion Processes. Second one is the size (number of elements) of the File Buffer. Third one is the search string and the last one is search path

After starting, the File Searcher Main Process creates (forks) child processes, which execute the Minion Processes and a pair of pipes for each minion to communicate with it. The File Searcher Main Process also assigns a unique minion key for each minion in creation. The key should be started with "minion" word and continued with a number up to the number of minion in the system, e.g. if there are three minions, then the keys are *minion1, minion2, minion3*. The File Searcher Main Process will also have a separate Controller Thread for each Minion Process to handle search requests.

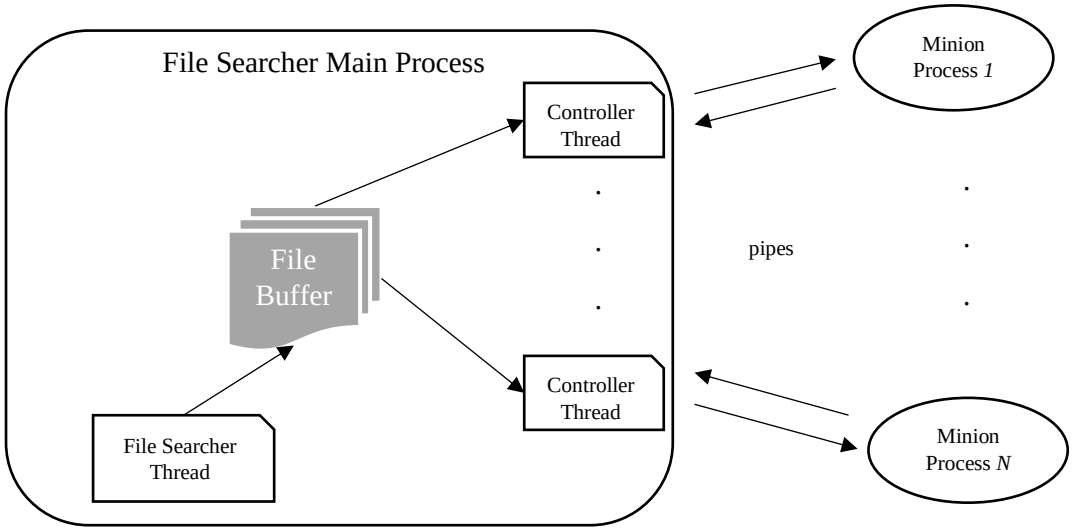

Figure 1. General schema of the file searcher system.

The File Searcher Thread starts its work immediately after the File Searcher Main Process is started, it does not have to wait for any minion to be available. Its main concern is the status of the File Buffer (whether it's full or has remaining space). This thread searches a directory provided from command line for *.txt files and feeds the found file information to File Buffer (The directory may contain different file types (extensions), and sub directories with *n* depth). File information to be held in the File Buffer is the path of txt files. As the File Buffer size fixed during the execution, it will need to be handled with a FIFO approach (You should understand that it's just an array of file info structures.) You should pay attention for correctly managing the buffer for concurrent accesses.

A Minion Process gets the information (path) of the files to be searched from the input pipe. (You may provide the search string via pipes or on the creation of Minion process). It opens the file and searches its content and writes the found string location (in means of line and column) to its output file. Name of the minion output file depends on the minion key e.g. *minion1.out, minion2.out, ..., minionN.out*. Each Minion Process reports back to the Main Process the found location (and any other relevant information) as an indication of successful task completion.

Controller threads handle the pipe communication between minion processes and main process. They consume the buffer and log the found information from minion processes. You should synchronize the usage of critical sources (log file, buffer) on controller threads.

The File Searcher Main Process also manages an output file named “searchlog.txt” in order to log its operations. This log file should be created from scratch at the start of each search request. Each line in this output file contains a log of the search operation completed, with the minion id, file path, found string location, e.g. minion1: /home/data/dirsearch/1.txt:1:25.

2.1. Execution of System

The program will be executed with four command line arguments.

./main <number of minions> <size of file buffer(# of elements)> <search string> <search path>

```
>gcc main.c -o main -Wall -ansi -lpthread
>gcc minion.c -o minion -Wall -ansi -lpthread
>./main 3 5 "at" /home/data/dirsearch
```

2.3. Minion Output File Format

Each minion will create an output file named *minion#.out (e.g. minion1.out, minionN.out)*. In each line, the found matches will be shown as *<minionkey>:<relative path of txt file>:<line index>:<column location>*. Please start the location indexes from 1 (First line and column index will be given as 1)

```
Search for "at" string on Minion1
---------------------
minion1: /home/data/dirsearch/1.txt:1:20
minion1: /home/data/dirsearch/1.txt:2:3
minion1: /home/data/dirsearch/1.txt:5:1
.
.
.
minion1: /home/data/dirsearch/sub3/sub31/ab.txt:1:11
.
.
.
```

2.4. Log File

Minions should send the information of a found string immediately to its controller thread, which logs the findings to the output file, *searchlog.txt.*. In each line, the found matches will be shown as *<minionkey>:<relative path of txt file>:<line index>:<column location>*. Please start the location indexes from 1 (First line and column index will be given as 1).

```
Log File
Search for "at" string
---------------------
minion1: /home/data/dirsearch/1.txt:1:20
minion2: /home/data/dirsearch/2.txt:3:10
minion2: /home/data/dirsearch/2.txt:3:12
minion1: /home/data/dirsearch/1.txt:2:3
```

```
minion1: /home/data/dirsearch/1.txt:5:1
minion3: /home/data/dirsearch/subfold1/a.txt:1:23
minion3: /home/data/dirsearch/subfold1/a.txt:1:52
.
.
.
```

2.5. Submission Format

<StudentID>.zip

src (*folder*)

- main.c
- main.h
- minion.c
- minion.h

3. Report

You should provide a report which gives a brief description of your solution and algorithm of your program.

4. Important Notes

- Please use understandable variable and function names. Respect to C naming conventions.
- Provide comments for each of your functions. Also provide comment lines on critical parts of your codes, if necessary.
- Use mutexes for synchronization, pipes for inter-thread communication and threads for parallelization.
- The described model does not contain all the implementation details. You should determine the communication protocol between the two processes, parameters to pass to threads and how to pass them. You may further enhance the model by adding new threads if you really think that it makes sense.
- You will use online submission system to submit your experiments. (https://submit.cs.hacettepe.edu.tr/). No other submission method (such as; CD or email) will be accepted. Late deliveries will not be accepted, either.
- Do not submit any file via e-mail related with this assignment.
- SAVE all your work until the assignment is graded.
- The assignment must be original, INDIVIDUAL work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms or source codes.
- You can ask your questions through course's Piazza group and you are supposed to be aware of everything discussed in the group.