Process Overseer

CAB403 Systems Programming

Semester 2, 2020

Due date: 2020-10-25 (Sunday week 13)

Weight: 40%

Group size: 1 - 3

Specification version: 1.01

Contents

1 Overview

A total of 40 marks are available for this assignment, representing 40% of your final grade. The specification is structured in such a way that you are only required to implement sections A and B to pass the assignment, provided your implementation is of sufficient quality. Subsequent sections represent additional, independent features of the assignment that will boost your marks.

Two programs are required: a server program "overseer" and a client program "controller", networked via BSD sockets (as covered in week 7). The overseer runs indefinitely, processing commands sent by controller clients. The controller only runs for an instant at a time; it is executed with varying arguments to issue commands to the overseer, then terminates.

The usage of the overeseer is shown below.

```
overseer <port>
```

The usage of the controller is shown below.

```
controller <address> <port> {[-o out_file] [-log log_file] [-t seconds]
<file> [arg...] | mem [pid] | memkill <percent>}
```

- `< >` angle brackets indicate required arguments.
- `[ ]` brackets indicate optional arguments.
- ... ellipses indicate an arbitrary quantity of arguments.
- `{ }` braces indicate required, mutually exclusive options, separated by pipes `|`. That is, one and only one of the following must be chosen:
  - `[-o out_file] [-log log_file] [-t seconds] <file> [arg...]`
  - `mem [pid]`
  - `memkill <percent>`

Table 1 provides an overview of the arguments and their associated sections.

Section	Argument	Description
A	`<address>`	overseer IP address
A	`<port>`	overseer port
A	`<file>`	the file to be executed
A	`[arg...]`	an arbitrary quantity of arguments passed to `file`
B	`--help`	prints usage message
B	`[-o out_file]`	redirects stdout and stderr of the executed `<file>` to `out_file`
B	`[-log log_file]`	redirects stdout of the overseer's management of `<file>` to `log_file`
D	`[-t seconds]`	specifies the timeout for SIGTERM to be sent
E	`mem [pid]`	get memory information
E	`memkill <percent>`	kill children if they use too much memory

Table 1: Controller arguments

2 A: Core (10 marks)

- The controller must attempt to establish a connection with the overseer via BSD sockets. If a connection could not be established, the controller must write to stderr:

  ```
  Could not connect to overseer at <ip address> <port>
  ```

- The controller must forward all arguments (other than the address and port) to the overseer.

- The overseer must accept incoming connections from the controller. If a connection is received it must write to stdout:

  ```
  %Y-%m-%d %H:%M:%S - connection received from <ip address>
  ```

- The requested file must be executed with all of its arguments. This must be achieved via `exec` (see man exec). However, `exec` will replace the process (so `fork` must be used here).

- The overseer must log its management of the executing process with timestamps of format `%Y-%m-%d %H:%M:%S`. Logging is written to stdout. Under normal circumstances, the below log is expected. Whenever a process terminates, its status code must be logged as below.

  ```
  %Y-%m-%d %H:%M:%S - attempting to execute <file> [arg...]
  %Y-%m-%d %H:%M:%S - <file> [arg...] has been executed with pid
  <pid>
  ```

```
%Y-%m-%d %H:%M:%S - <pid> has terminated with status code <status
code>
```

- If the file could not be executed this must be logged as below.

```
%Y-%m-%d %H:%M:%S - attempting to execute <file> [arg...]
%Y-%m-%d %H:%M:%S - could not execute <file> [arg...]
```

3 B: Core extension (4 marks)

- If the first argument to the controller is `--help` then the following usage message must be written to stdout before instantly terminating.

```
Usage: controller <address> <port> {[-o out_file] [-log log_file]
[-t seconds] <file> [arg...] | mem [pid] | memkill <percent>}
```

- If the first argument is not `--help` then the controller must validate argument usage. This includes checking the correct types for each argument and that the arguments are in the correct order. Invalid usage must cause the controller to write the same usage message as above, but to stderr.
- The overseer must obey the optional argument `-log log_file` to redirect the logging of its management of the process to `log_file`.
- If the optional argument `-o out_file` is used, the process's stdout and stderr handles must be redirected to `out_file`.
- Output redirection can be implemented using `dup2` (see man dup).

4 C: Thread pool (5 marks)

- The overseer must be implemented as a thread pool using POSIX Threads. All threads must be started when the overseer is launched. The size of the thread pool must be fixed at 5.
- When the overseer accepts a connection, it must append to a global queue (implemented as a linked list) the request from the controller.
- After appending the request to the queue, one thread must be signalled, via a condition variable, to process the next request in the queue. Handling a request involves everything required by the overseer, including executing the process, redirecting output, logging, and signalling (if section D is implemented).
- Mutexes must be used to synchronize access to the queue, avoiding all race conditions.

- When a thread finishes handling a request (either because the process could not be executed or because it has terminated) then the thread must wait for the next condition variable signal.
- Any kind of execution and management of a child process must be handled in a pthread. The main thread must only serve to receive connections and pass them off to threads. This rule extends to other sections if this section has been implemented.

5 D: Signals (4 marks)

- After a subordinate process has been running for 10 seconds, the overseer must send a `SIGTERM`. This timeout can be customized via the `-t` argument. When a `SIGTERM` is sent, this action is written to the log as below. Normally the process will clean itself up nicely and terminate.

```
%Y-%m-%d %H:%M:%S - sent SIGTERM to <pid>
%Y-%m-%d %H:%M:%S - <pid> has terminated with status code <status
code>
```

- If the process has not terminated after 5 seconds since sending the `SIGTERM`, it must be forcefully killed via `SIGKILL`. This must also be written to the log as below.

```
%Y-%m-%d %H:%M:%S - sent SIGTERM to <pid>
%Y-%m-%d %H:%M:%S - sent SIGKILL to <pid>
```

- The overseer must handle `SIGINT` by cleaning up all resources, including dynamic memory, sockets, and children, which must be all instantly killed via `SIGKILL`. Then, the overseer must terminate.

6 E: Memory regulation (8 marks)

The process information pseudo-filesystem proc enables exploration of various kinds of information associated with any given process. Each process has a directory under this file system located at `/proc/[pid]`. One such file is `/proc/[pid]/maps` which lists the mapped memory regions of `pid` at any moment in time. You can find the detailed description of this file by searching for `/proc/[pid]/maps` in the man page. By summing the amount of memory mapped at each entry, one can determine the amount of memory a process is using at any time. Note that you must only count the entries with an inode of 0, as the others are memory-mapped files.

- Every second, for each running process, an entry must be appended to a global linked list where each entry contains the process's PID, the current time, and the memory (in bytes) used by the process. If you have implemented section C, this necessarily involves an additional mutex.

- At any point in time, the controller can request this memory information via one of two special commands:
  - `mem` returns to the controller a list of all running child processes, including their last recorded memory usage. This report must be written to the controller's stdout. Each line of this report takes the below format.

    ```
    <pid> <bytes> <file> [arg...]
    ```

  - `mem <pid>` returns to the controller the entire memory usage history for the specified process in chronological order. This report must be written to the controller's stdout. Each line of this report takes the below format.

    ```
    %Y-%m-%d %H:%M:%S <bytes>
    ```

- At any point in time, the controller can issue a third kind of special request `memkill <percent>` where all processes using more than a certain percentage of the system's memory must be terminated with `SIGKILL`. The percent is expressed without the % symbol. For example `memkill 12.5` would issue a `SIGKILL` to all processes currently using more than 12.5% of the system memory. The total usable main memory size must be determined using the sysinfo system call.

7 Build automation (1 mark)

You must write a Makefile that enables the marker to compile your programs by simply running 'Make' in the directory of your project. A Makefile also makes your development more efficient.

8 Implementation quality (8 marks)

Code quality and overall program design is important; this includes but is not limited to:

- Program must be reliable (run time errors, segfaults, deadlocks etc must be non-existent).
- Code must be non-redundant.
- Code must be well-commented.
- Busy-waiting must be avoided.

- Resources are well-managed and freed. Use of dynamic and static memory where appropriate.
- Network byte order must be used when transmitting data via socket communication (this only applies if you are sending non-character data).

9 Submission

First, fill in the `statements.txt` file (see Blackboard). This file is a statement of completeness and contributions, where you will describe which parts of the assignment your group has completed and how each of your group members contributed to the project.

Then, create a .zip format archive (this can be created in Linux with the `zip` command; type `man zip` for details) containing your submission along with this file. The .zip must not contain any subdirectories. The "extract here" command executed on the .zip must extract the following files to the current directory:

- `Makefile`
- `statements.txt`
- source code including any header files

Running `make` in the current directory must then generate two binaries in the current directory (there should be no directories generated):

- `overseer`
- `controller`

Running `make clean` must delete any compiled binaries or object files (cleaning the project directory of any files generated by running `make`.)