Job Submission System

Software Design Document

Xiangshuai Gao, Hyejun Jeong

CSE 376 HW4

05/18/2020

1. Requirements

The general goal is to develop a client/server system for job submissions: a client that can submit jobs to be executed with various criteria; and the server that will process these jobs asynchronously and report their status.

Server

The server will have a number of configuration options that you can pass on the command line:

1. The maximum number of jobs that can be concurrently run.
2. The server will track resources used by each running job (a process): time and CPU cycles consumed, memory used, etc. The server will be able to take action on a process that exceeds these capabilities: kill a job, stop it, and change it to a different priority, etc.
3. The server will be able to report back to the client the status of one or more jobs: list one or more jobs, resources they're using, are they running or not (and why not), and the output these jobs produced on stdout/stderr.

Client

The client will be able to issue commands to the server to list one or more jobs, check their status, tell the server to suspend/kill a process or change its capabilities, retrieve status/error codes, etc.

The client could submit new jobs. New jobs will be described as a vector of argv[] and envp[] strings, along with an additional data structure that describes the restrictions on the job to run (max time to run, max resources to consume, priority, etc.).

Client/Server Interaction

The client and server should communicate via a Unix pipe (domain socket). This is a special file you create on the file system that has two communication channels. The server will create the pipe and listen for commands on it (e.g., "/home/jdoe/.hw4server_control"). The client could write(2) to the pipe and can read(2) back from it. When one side writes to a pipe, the other side can read back from it.

You will have to design a small protocol for communicating over the pipe.

2. Design

2.1. Protocol Design

2.1.1. Packet Structure

We used a slightly different protocol than the ones described in the HW4.pdf. Our protocol consists of a command byte, which has a value of 1,2,3. 1 is for submitting new jobs, 2 is for listing jobs, 3 is for killing jobs as per the HW4.pdf. The next 4 bytes are the job message size, which describes the length of the buffer containing all the information about the job. The following 24 bytes are divided into four each holds the values for max memory, max CPU time, max priority, envp size, argv size, argc, respectively. The rest of the bytes after that are consecutive strings of argv and envp each terminated by a null byte. For the client side, no protocol is used. Instead, the server sends back plaint text information about the status of a client's request.

2.2. Data Structure

Client LinkedList

All the client connections are stored inside a Linked List structure. Within the LinkedList, there is the Client structure, which is stores all the information about the client. Each client object holds the client file descriptor as well as a Linked List of jobs. A new client connection will be inserted into this Linked List when the connection is accepted. The client that closes the connection will be removed when the connection is terminated.

Job LinkedList

Job LinkedList is another LinkedList inside the Client structure. Job structure describes the job submitted by the client such as the max memory, max CPU time, argc count, argv size, and envp size. These structures are where the job from the receiving buffer will be deserialized into. Argv and envp are treated as consecutive strings in the structure terminated by null bytes. Besides the Job structure, this LinkedList node also stores the exit status of the client, what it receives from the job's stdout and stdin, its job pid, as well as the pipe file descriptors used for reading the outputs from the job.

3. Implementation

3.1. Server

The server runs with the option -j [max jobs], which tells the server how many max jobs can be run at once. The default is 10 if this option is omitted.

For implementing a multiclient server, we are using the poll function. No threads and forks are used for this purpose. The server runs in an infinite loop and will hang until a new client connects or an existing client sends something. It will then process these requests and rises and repeat.

For new job submission, the server deserializes the receiving packet and creates a LinkedList job struct from it. It then uses fork(2) to fork a child process to begin the job. The child process checks the max CPU time, max memory, max priority, and sets them accordingly using the rlimit(2) system calls and setpriority(2) system calls.

Each terminated job is reaped by the parent server upon completion. When a client sends a list request, it will print all the jobs for that specific client. If a job terminated upon completion, the list request will display that job as terminated and print out the output of that job. For killing a job, we sing the kill system call. After killing a job, its job status will say killed. The output of this job will not be kept in this case and will say N/A. If a job is still running, it will display running as its status and N/A as its output since it is still running.

3.2. Client

After the Client runs, the user can enter the command, either submit, list, kill, or exit. In a infinite loop, the program will prompt the user to enter the command. If the command is not one of those four, the program will print an error message saying that the command is not found.

In the client side, the content of the buffer will be set. The client sends a buffer, which will be deserialized on server side and stored as a packet/job struct. The buffer is prepared for each command as below.

Job Structure for Submit

[1 byte]	command type number: 1
[4 bytes]	total length of the packet
[4 bytes]	max memory to consume in bytes
[4 bytes]	max time to run in seconds
[4 bytes]	priority
[4 bytes]	length of envp
[4 bytes]	length of argv
[4 bytes]	argc
[size varies in bytes]	envp[] (char array terminated by null)
[size varies in bytes]	argv (char array terminated by null)

Job Structure for List

[1 byte]	command type number: 2

Job Structure for Kill

[1 byte]	command type number: 3
[4 bytes]	jobpid to kill

For a new job submission, the program will prompt the user to enter three inputs: the max memory to consume in bytes, the max time to run in seconds, and the priority of the job. Each user input is converted to integer value using atoi. These values are stored in the buffer in order as described above. Argv and envp are the array of strings, delimited by null character.

To list all the jobs, the client just sends a byte, 2, representing the list command type.

For killing a job, the program will also prompt the user to enter the jobpid. This jobpid received by the user is converted to integer, copied to the buffer, and sent to the server.

After the buffer preparation as command type, the client sends the buffer using send. Then the client will receive the server's response accordingly.