SJSU CS 149 HW1 SPRING 2021

REMINDER: Each homework is individual. "Every single byte must come from you." Cut&paste from others is not allowed. Keep your answer and source code to yourself only - never post or share them to any site in any way.

[Type your answer. Hand-written answer is not acceptable.]
[Replace YourName and L3SID with your name and last three digit of your student ID, respectively.]

1. (20 pts) Given the following program.
a. (10 pts) Use a software to draw the space-time diagram (similar to the ones for fork2 and fork3 in slides) to justify your answer. Any software is fine. Hand-drawing receives no credit.
b. (10 pts) At the end of the program execution, what is the total number of processes (including the initial parent process)? List **all** values of `i` before each process terminates (and label `i`'s value in the space-time diagram).

```
int main() {
      int i = 1;
      if (fork() == 0) {
            ++i;
            if (fork() == 0)
                  i = 4;
            else {
                  ++i;
                  fork();
            }
      } else {
            i = 7;
            if (fork() != 0)
                  ++i;
      }
      if (i > 5) {
            fork();
            i = i + 2;
      }
      return 0;
}
```

2. (80 pts) [programming question]
Design a C program to serve as a shell command-line interface that accepts user commands and then executes each command in a separate process. Your implementation will support one form of inter-process communication – shared memory. HW1 can be completed on Linux or Linux VM.

A shell command-line interface gives the user a prompt, after which a user enters the next command. The format of the prompt is

`FirstName-L3SID>`

where L3SID is the last three digits of your student ID. Assuming your first name is `demo` and L3SID is `123`, the example below illustrates the prompt (`demo-123>`) and the user's next command: `cat prog.c` (which displays the file `prog.c` on the terminal using the UNIX `cat` command).

```
demo-123> cat prog.c
```

The shell process first prints the prompt, reads what the user enters on the command line (in the above case, `cat prog.c`), and then creates a separate child process that performs the command. Unless otherwise specified, the shell (parent) process waits for the child (executed in the foreground) to exit before printing the next prompt and continuing.

However, Linux shells typically also allow the child process to run in the background, or concurrently. To accomplish this, we add an ampersand (`&`) at the end of the command line. Thus, if we rewrite the above command as

```
demo-123> cat prog.c &
```

the shell (parent) and child processes will run concurrently. In both cases the separate child process is created using the `fork()` system call, and the user's command is executed by using one of the system calls in the `exec()` family. For any command running in the background, the shell (parent) process does not wait for the completion of the child process; the shell prints the prompt and reads the next command immediately.

A C program that provides the general operations of a command-line shell is as follows (you can copy this code).

```
#include <stdio.h>
#define MAXLINE 80 /* The maximum length of a command line */
#define MAXARGS 40 /* The maximum # of tokens in command line */
int main(void)
{
  char cmdline[MAXLINE];
  char *args[MAXARGS];
  …
  printf("CS149 Spring 2021 Shellshm from FirstName LastName\n"); /* replace w/ name */
  while (1) {
    printf("FirstName-L3SID>"); /* prompt- replace FirstName and L3SID */
    fflush(stdout);
    /* (1) shell - read user input (stdin) to cmdline
     * (2) shell - parse tokens from cmdline to args[]
     * (3) shell - if user enters exit, terminate the shell (parent)
     * (4) shell - fork a child process using fork()
     * (5) child - call execvp()
     * (6) shell – if no & at the end of cmdline, call wait() to wait for the termination of the child, else do nothing
     * (7) shell - go back to (1)
     */
  }
  return 0;
}
```

Executing commands in a child process

To read a line from the terminal (i.e., stdin), you may utilize fgets(3), readline(3), read(2), or any other APIs. To extract tokens from a string, you may use strtok(3), strtok_r(3), strsep(3), or any other APIs, or write your own parsing routine.

As an example, if the user enters the command "`ps -af`" at the prompt, the values stored in the `args` array are:

```
args[0] = "ps"
args[1] = "-af"
args[2] = NULL
```

Each string is null terminated, and the last entry in `args[]` must be NULL. This `args` array will be passed to the `execvp()` function, which has the following prototype:

```
execvp(char *command, char *params[]);
```

where `command` represents the command to be performed and `params` stores the parameters to this command. For Q2, the `execvp()` function should be invoked as `execvp(args[0], args)`. Before invoking `execvp()`, check whether the user included an `&` at the tail end of command line to determine whether the shell (parent) process is to wait for the child to exit. The character `&`, if it exists at the tail end of a command line, is meaningful to the shell only and is NOT an argument of the command (i.e., `&` is not a token in `args[]` when invoking `execvp()`).

When any system call failed (e.g., `fork()`, `execvp()`, etc.), the shell should print out error messages, output the prompt and accept the next command from user.

Your shell must parse the command line, whatever that command line is, and then execute the command specified.

Make sure for each invocation the program always prints out "`CS149 Spring 2021 …`" only once. Screenshots must include "`CS149 Spring 2021 …`" from the program. Follow the format for the prompt which includes FirstName and L3SID.

Elapsed time of command

After child process running in the foreground exits, your shell prints out the total elapsed time (in seconds, example in bold) to run the child before printing the next prompt. For example,

```
demo-123> cat prog.c
… content of prog.c …
(elapsed time: 0.067527 seconds)
demo-123>
```

The general idea is to use shared memory between the parent and child process. The parent should setup the shared memory first. Before calling `execvp()`, the child process calls `gettimeofday()` as the starting time of the specified command and deposits the starting time to the shared memory. After returning from `wait()`, the parent calls `gettimeofday()` as the ending time of the specified command. After retrieving the starting time from the shared memory, the parent calculates and prints out the elapsed time to stdout.

The `gettimeofday()` function records the current timestamp. This function is passed a pointer to a struct `timeval` object, which contains two members: `tv_sec` and `tv_usec`. These represent the number of elapsed seconds and microseconds since January 1, 1970 (known as the UNIX EPOCH). The following code sample illustrates how this function can be used:

```
struct timeval current;
gettimeofday(&current, NULL);
// current.tv_sec represents seconds
// current.tv_usec represents microseconds
```

The delta between the starting and ending time is the elapsed time. If the shell calls `printf()` to print the elapse time, the format of the elapsed time in `printf()` format (in units of second) should be `%ld.%06ld`.

The shell does not print the elapsed time for any child running in the background.

Sample source code from the textbook

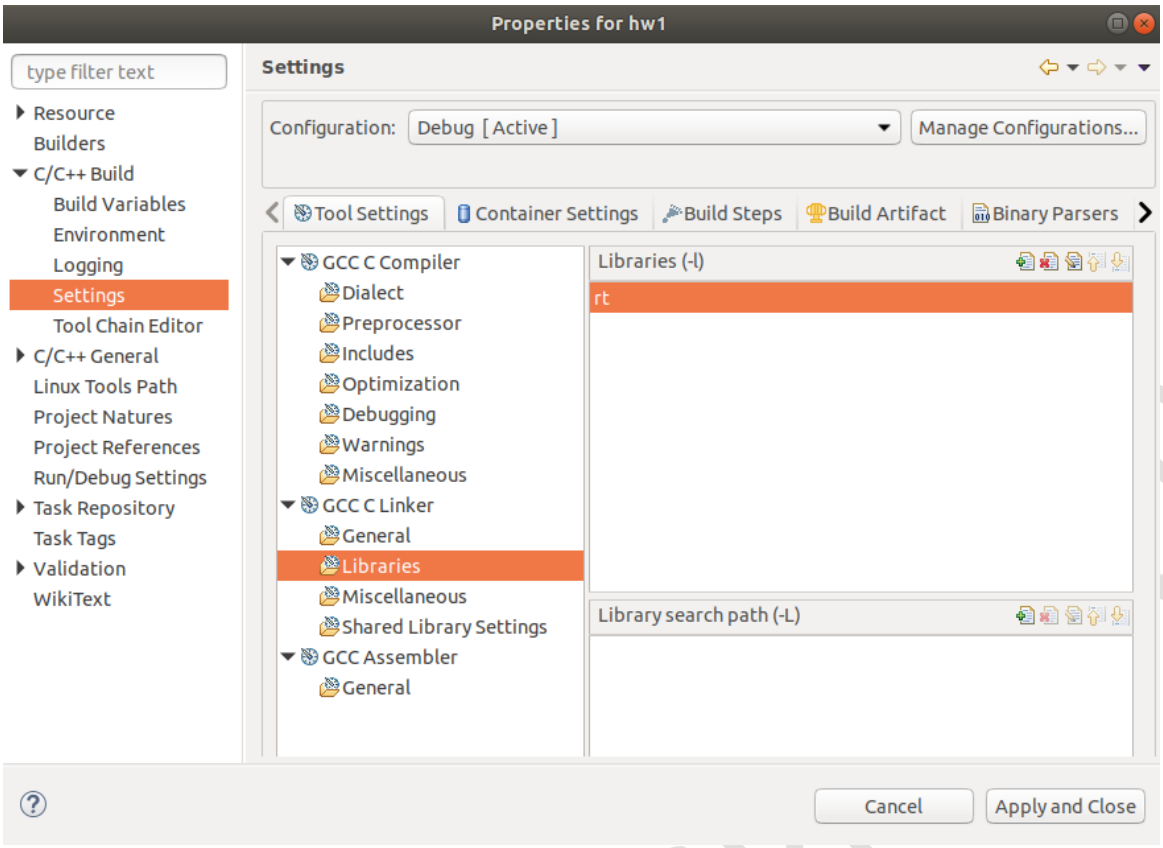
under ch3 directory: shm-posix-consumer.c, shm-posix-producer.c

Compilation and Linking

Various shared memory related APIs are not in the standard C library. From a real shell in a Linux terminal, you can compile your program with "`gcc -o shellshm shellshm.c -lrt`", and execute the program with "`./shellshm`".

How to add linker option `-lrt` to a project in Eclipse?

Project -> properties -> C/C++ Build -> Settings -> GCC C Linker, under Libraries (-l), add rt



Reminders

1. You must utilize array for various data structure and utilize loop to remove repeating code. Any repetitive variable declaration and/or code are subject to deduction.

2. Best practice:

a. include necessary header files only.

b. remove any warning messages in compilation.

c. error handling: always checks the return code from any API.

What to do next

After you finish debugging your code,

a. (15 pts) Do the followings in a "terminal" on Linux VM:

- In a terminal running a real shell, bring up your shell: `./shellshm`
- In your shell, invalid command (print error msg): `xyz`
- In your shell, empty command line (move to next line):
- In your shell, sleep 5 seconds (to check elapsed time): `sleep 5`
- In your shell, list the directory /usr: `ls -c /usr`
- In your shell, get current date: `date`
- In your shell, sleep 789 seconds in the background: `sleep 789 &`
- In your shell, show processes: `ps -f`
- In your shell, exit your shell: `exit`
- Now back to a real shell, show processes: `ps -f`

- Take screenshot of the entire sequence (which includes "`CS149 Spring 2021 …`" from your program). Multiple screenshots are OK.

b. (10 pts) In a, the output from two executions of "`ps -f`" should include the same process "`sleep 789`" with identical PID but with different parent PID (PPID).

- (5 pts) Explain the reason, by starting with "The child process becomes a(n) _____ process because ….."
- (5 pts) Identify which process is the new parent process of "`sleep 789`". This can be done by invoking the following command in the real shell from a terminal (where `NNN` is the new PPID of "`sleep 789`").

  ```
  ps -p NNN
  ```

  Include a screenshot of the execution of "`ps -p NNN`".

c. (5 pts) If your shell never waits for the termination of any user-entered command (regardless whether the tail of the command includes "`&`" or not), what would happen to these processes (corresponding to user-entered commands) after these processes call `exit()`, assuming your shell is still up and running? Your answer must start with "these processes become _____ processes because …"

d. (50 pts) source code as a separate file (shell: 35 pts, shared memory/elapsed time: 15 pts).

====

Submit the following files as individual files (do not zip them together):

- `CS149_HW1_YourName_L3SID` (.pdf, .doc, or .docx), which includes
  - Q1: answers and justification
  - Q2: answers to a, b, and c. Screenshots that are not readable, or screenshot without "`CS149 Spring 2021 …`" from the program, will receive 0 point for the entire homework.
- `shellshm_YourName_L3SID.c` Ident your source code and include comments. Also follow the exact format for the prompt as specified. Any submission without the proper FirstName and L3SID in the prompt will receive 0 point for the entire homework.

The ISA and/or instructor leave feedback to your homework as comments and/or annotated comment. To access annotated comments, click "view feedback" button. For details, see the following URL:

https://guides.instructure.com/m/4212/l/352349-how-do-i-view-annotation-feedback-comments-from-my-instructor-directly-in-my-assignment-submission