Software Tools and Systems Programming — Final Exam

Midterm test

April 19th, 2024 – Duration: 3 hours

Total: 37 points + 1 bonus pts.

5 Problems

6 Questions + 1 bonus qn.

NAME	
Student Nbr	
Lecture Session	

Instructions

- Do **not** turn this page until you have received the signal to start.
- Please fill out the identification above and read these instructions.
- No aids are allowed.
- Please do not dettach any pages.
- Do not write on the QR code at the top of the pages.
- Lecture session is either: "Morning" (02) or "Afternoon" (01).

G o o d L u c k ! ! !

Problems

Problem #1 [5 pts.]

Write a shell script that attempts to download a web-page from the Internet. The script will attempt to download the web-page using the `curl` command available in the shell.

The script will finish when the download of the file has been successful: it may occur that the transmission of the file is interrupted by different reasons (bad connection, server unresponsive, etc.), in that case the curl command will fail.

The `curl` command is complaint with the error code standards in *nix OS. If there was an error during the downloading, then your script will re-attempt to download the file for a maximum number of times. The script will accept two command line arguments, the first one being the URL of the web-page, e.g. "http://www.myURL.com/index.html"; and the second one the number of attempts the script will attempt to download the file.

The script should check for the proper number of arguments: - the URL is a mandatory argument - the maximum number of attempts is an optional argument, and when not specified it should default to 5.

The name of the downloaded file should be "`file_transfer--X`", where "`X`" represents the current downloading attempt number. It is OK if the script generates and keeps previous attempts of incomplete/unsuccessful downloads.

You may assume that the URL is correct and provided as needed.

This is a bit of the snippet of the curl command documentation:

```
curl(1)                         curl Manual                         curl(1)

NAME

curl - transfer a URL

SYNOPSIS

curl [options / URLs]
```

```
DESCRIPTION
curl  is  a  tool for transferring data from or to a server using URLs. It sup-
ports these protocols: DICT, FILE, FTP, FTPS,  GOPHER,  GOPHERS,  HTTP,  HTTPS,
IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP, RTMPS, RTSP, SCP, SFTP, SMB,
SMBS, SMTP, SMTPS, TELNET, TFTP, WS and WSS.
. . .
-o, --output <file>
 Write output to the given file instead of stdout.

Example:
 curl "http://www.example.com" -o "file_downloaded.txt"
```

Examples:

```
# CLA checks

$ ./file_transfer.sh
This program requires at least one argument indicating the URL to download.
Usage: ./file_trasnfer.sh <file_url> [max_attempts]

$ ./file_transfer.sh http://www.myURL.com/index.html 12 34
This program requires at least one argument indicating the URL to download,
and a maximum of 2.
Usage: ./file_trasnfer.sh <file_url> [max_attempts]

# Valid use cases
$ ./file_transfer.sh  http://www.myURL.com/index.html
Attempt #1 to download  http://www.myURL.com/index.html
Download sucessful!
File saved in file_transfered_1

$ ./file_transfer.sh  http://www.myURL.com/index.html  7
Attempt #1 to download  http://www.myURL.com/index.html
Download failed! ...
```

```
Attempt #2 to download  http://www.myURL.com/index.html
Download failed! ...
Attempt #3 to download  http://www.myURL.com/index.html
Download failed! ...
Attempt #4 to download  http://www.myURL.com/index.html
Download failed! ...
Attempt #5 to download  http://www.myURL.com/index.html
Download failed! ...
Attempt #6 to download  http://www.myURL.com/index.html
Download failed! ...
Attempt #7 to download  http://www.myURL.com/index.html
Download failed! ...

$
```

Problem #2 [5 pts]

Write a C program that demonstrates the creation of child processes and handles their cleanup. The program should achieve the following:

- The main process (parent) should create a fixed number of child processes as indicated by a command line argument (CLA); if not CLA is passed to the program then it will generate 5 child processes.
- Each child process should print it child's number and PID, and sleep for a second (to simulate some work being done).
- Ensure that no zombie processes remain after the child processes exit.
- The parent process should wait for all child processes to finish before exiting.
- If the parent process receives a termination signal (e.g., SIGINT), it should gracefully terminate all child processes and then exit.

Example Output

```
Parent process (PID 1234) created 5 child processes.
    Child 1 (PID 5678) started.
    Child 2 (PID 5679) started.
    Child 3 (PID 5680) started.
    Child 4 (PID 5681) started.
    Child 5 (PID 5682) started.
    Child 1 (PID 5678) completed its work.
    Child 2 (PID 5679) completed its work.
    Child 3 (PID 5680) completed its work.
    Child 4 (PID 5681) completed its work.
    Child 5 (PID 5682) completed its work.
All child processes terminated gracefully.
Parent process (PID 1234) exiting.
```

Problem #3 [5 pts]

Consider a program that creates two child processes. Each child process should send its own PID to the other process.

(a) Specify all the possible IPC methods that could be used for implementing this?

(b) Implement a C program using one of these methods, so that the output will look like this:

```
Parent 338357 waiting for child processes (338358,338359) to finish...
Child 2 received PID from Child 1: 338358
Child 1 received PID from Child 2: 338359
```

Implement this program taking into consideration the error checks in each appropriate instance. Consider strategies to eliminate the chance of generating orphans or zombies processes.

Problem #4 [5 pts]

Write another C program achieving the exactly the same as the program you wrote in Problem #3, i.e. child processes communicating their corresponding PIDs, but this time employing a different IPC method than the one you used to solve Problem #3.

Implement this program taking into consideration the error checks in each appropriate instance. Consider strategies to eliminate the chance of generating orphans or zombies processes.

Problem #5 [5 pts]

Write a C program that generates a process which remains active in the background independently of how the program is run and keeps printing the following information to standard output every 10 minutes:

```
>> rep.nbr.: X
Date and Time:  XXXXXXXXXXXXXXXXXXXXXX
>> Users in the system:
USER TERMINAL DATE HH:MM
>> You are current directory is XXXXXXXXXXXXXXXXXXXXX
```

```
~/marcelo> ./background_reporter

  ------------------------------------------------------------------
  >> rep.nbr.: 0
  Date and Time: Fri Apr 19 19:00:00 2024

  >> Users in the system:
  marcelo tty 2024-04-01  23:09
  marcelo pts/0 2024-04-13  00:29
  paco   pts/1 2024-04-19  00:00
  albert   tty3  2024-04-13  11:01

  >> You are current directory is /home/users/marcelo
  ------------------------------------------------------------------

~marcelo/> ls

codes/  Documents/ Desktop/ Downloads/

~marcelo/>

  ------------------------------------------------------------------
  >> rep.nbr.: 1
  Date and Time: Fri Apr 19 19:10:00 2024

  >> Users in the system:
  marcelo tty 2024-04-01  23:09
  paco   pts/1 2024-04-19  00:00

  >> You are current directory is /home/users/marcelo/Desktop
  ------------------------------------------------------------------

~marcelo/>
```

For printing the required information you can either use system calls or external commands from the shell to be run by the C program. If you use this second approach be careful in which way you decide to do this and explain your logic in the code.

This background *reporter* process will continue running...

Questions

Please answer the following questions providing as much detail as possible, as well as, clearly justifying your responses:

Question #1 [2 pts.] Consider the program you wrote in *Problem #5*,

(a) Is it possible to end the execution of the reporter in background process?
If you are answer is yes, how?
If your answer is not, why not?

(b) If you answered ”yes” before, are there any changes to the program that may prevent us from possibly ending the program? Explain.
If you answered ”no” before, are there any changes to the program that may allow us to end the program execution? Explain.

Question #2 [2 pts.]

(a) Draw the *process state diagram* illustrating the different **states** a process can go through in an operating system, along with their transitions.

(b) Describe the different states and transitions in the diagram.

(c) Provide an example for each of them.

Question #3 [2 pts.]

(a) What is the *Process Control Block*? Provide details about what is it used for?

(b) What is a **race condition**? Provide an example.

(c) How are these two previous elements related to *threads*?

Question #4 [2 pts.]

(a) What is a *socket*?

(b) What are sockets used for?

(c) Enumerate and describe the different stages in their typical use.

Question #5 [2 pts.]

(a) What is the *shell*?

(b) What is the *kernel* of an OS?

(c) What are *system calls*?

(d) Provide one example related to each of these previous elements.

Question #6 [2 pts.]

(a) What is *I/O multiplexing*? Why is it needed/used?

(b) Describe the different *I/O Models*?

(c) Compare the fundamental I/O multiplexing *system calls* discussed in the course?

Bonus

Bonus mark [1 pt.]

Tell us what was/were the most *interesting topic(s)* and most *useful tool(s)* you learned in CSCB09-w'24 and **why**?

Scratch/Extra space

Cheat Sheet

Make – Automatic variables

$@	The file name of the target of the rule.
$*	The file name of the target without the file extension.
$<	The name of the first prerequisite.
$?	The names of all the prerequisites that are newer than the target, with spaces between them.
$^	The names of all the prerequisites, with spaces between them. Discard duplicates.
$+	Similar to $^, but includes duplicates.

Special Shell Variables

`$0`	filename of the current script
`$n`	*n*-th argument
`$#`	number of arguments
`$@`	list of all arguments
`$*`	list of all arguments when in quotes
`$?`	exit status
`$$`	process ID of the current shell
`$!`	process number of the last background command

Some Useful C Functions

```
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *argv[])
int fclose(FILE *stream)
char *fgets(char *s, int n, FILE *stream)
pid_t fork(void)
int pipe (int filedes[2])
FILE *fopen(const char *file, const char *mode)
```

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);
int pthread_join(pthread_t thread, void **value_ptr);
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void pthread_exit(void *value_ptr);
```

```
int sigaction(int sig, const struct sigaction *act, struct sigaction *oact)
int sigaddset(sigset_t *, int);
int sigdelset(sigset_t *, int);
int sigemptyset(sigset_t *);
int sigfillset(sigset_t *);
unsigned sleep(unsigned seconds);
int sigismember(const sigset_t *, int);
int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
```

struct sigaction:

- void (*sa_handler)(int) *Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.*
- sigset_t sa_mask – *Set of signals to be blocked during execution of the signal handling function.*
- int sa_flags – *Special flags.*
- void (*sa_sigaction)(int, siginfo_t *, void *) – *Pointer to a signal-catching function.*

int fprintf(FILE * restrict stream, const char * restrict format, ...);	
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);	
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);	
pid_t getpid(void);	
pid_t getppid(void);	
int kill(int pid, int signo)	
void *malloc(size_t size);	
int open(const char *path, int oflag)	
int scanf(const char *restrict format, ...);	scans input according to a format
char *strncat(char *dest, const char *src, size_t n)	concatenate strings
char *strstr(const char s1, const char *s2);	locates the first occurrence of s2 in s1
int strcmp(const char *s1, const char *s2);	
int strncmp(const char *s1, const char *s2, size_t n)	compare strings
char *strncpy(char *dest, const char *src, size_t n)	copy strings
size_t strcspn(const char *s, const char *charset);	find offset of first character match
size_t strlen(const char *s)	string length
char *strpbrk(const char *s, const char *charset);	find characters in string
char *strrchr(const char *s, int c);	locate last occurrence of character in string
size_t strspn(const char *s, const char *charset);	find offset of first non-matching character
char *strtok(char *restrict str, const char *restrict sep);	string tokens
long strtol(const char *restrict str, char **restrict endptr, int base);	convert to long
int wait(int *status); int waitpid(int pid, int *stat, int options)	/* options = 0 or WNOHANG*/

Useful Macros

```
  WIFEXITED(status) WEXITSTATUS(status) WIFSIGNALED(status) WTERMSIG(status) WIFSTOPPED(status)
WSTOPSIG(status)
```