Lab 3
Processes and File I/O

CSCI 4061
Introduction to Operating Systems

Sept 25, 2023

Overview

- Processes
  - wait
  - exec
- File I/O
- Exercise

Division of points for lab:

- Total points: 1
- Lab submission: 0.75
- Lab attendance: 0.25

Processes

Terminal

```
$ make lab2_p3
Value of x in parent process [864408] = [4]
Value of x in child [864409] with parent [864408] = [2]
Parent of child process [864409] = 3470

$ make lab2_p3
Value of x in parent process [45641] = [4]
Value of x in child [45642] with parent [45641] = [2]
Parent of child process [45642] = 1
```

Processes

Terminal

```
$ make lab2_p3
Value of x in parent process [864408] = [4]
Value of x in child [864409] with parent [864408] = [2]
Parent of child process [864409] = 3470
$ make lab2_p3
Value of x in parent process [45641] = [4]
Value of x in child [45642] with parent [45641] = [2]
Parent of child process [45642] = 1
```

Why is the parent process reported differently?
systemd / init process (usually pid = 1, however there can be multiple systemd in user mode)
$\Longrightarrow$ wait()

Processes

wait()

Parent process waits for

- a specific child to complete (p1.c)
- all its children to complete (p4.c)

In last lab, parent process exited before child causing a switch of parent to a system process.

Syntax

```
#include <sys/wait.h>
pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
// returns process ID of child
// returns -1 on error and sets errno
```

Processes

Sample code in sample folder(p1.c)

```
    else{
        x += 3;
        printf("Value of x in parent process [%d] = [%d]\n",
                                        getpid(), x);
        // waits for pid child to complete execution
        // waitpid(pid, NULL, 0);

        // Just check if pid child has terminated and return
        // similar to no wait if child not terminated
        // int status;
        // waitpid(pid, &status, WNOHANG);

        // Wait for any child
        // wait(NULL);
    }
```

```
# Try uncommenting each code snippet of wait
$ make lab3_p1
```

Processes

exec()

- Overlays the calling process image with a new one
- Child executes a new program and parent continues with the existing one
- Six variants of exec:
  - execl, execle, execlp - explicit list of arguments passed to exec call (compile time)
  - execv, execve, execvp - an array of arguments passed to exec call (run-time)

Syntax

```
#include <unistd.h>
// execl last parameter must be NULL
int execl(const char *path, const char *arg0, ...,
                            const char *argn, char *(0));
int execv(const char *path, char *const arg[]);
```

Processes

Terminal (p2.c, p3.c)

The programs demonstrate execution of *ls* command on parent and child using different parameters. The parent waits for child to complete.

```
# execl
$ make lab3_p2
# execv
$ make lab3_p3
```

File I/O

UNIX File interface

- Everything is a file in UNIX and its derivatives
- Uniform interface provided for I/O
- System calls - `open, close, read, write, lseek`
- File descriptor - unique identifier for a file
  - File descriptor $\rightarrow$ Process file descriptor table $\rightarrow$ System file table $\rightarrow$ Inode table

File I/O

Standard File I/O

- Wrappers around system calls with more flexibility
- I/O calls - `fopen, fscanf, fprintf`
  `fread, fwrite, fgets, fputs`
  `fclose, ftell, fseek`
- FILE structure instead of File descriptor
- FILE structure - {buffer, file descriptor}
- Use standard file I/O in your project

Standard File I/O API (p4.c)

```
#include <stdio.h>
// open file- mode:r, w, a, r+, w+, a+
FILE *fopen(const char *filename, const char *mode);

// read from file
size_t fread(void *buffer, int datatype_size,
                    int num_bytes_to_read, FILE *fp);
int fscanf(FILE *fp, const char *format, ...);
char *fgets(char *buffer, int num_bytes_to_read, FILE *fp);

// write to file
size_t fwrite(void *buffer, int datatype_size,
                    int num_bytes_to_write, FILE *fp);
int fprintf(FILE *fp, const char *format, ...);
int fputs(char *buffer, FILE *fp);
```

File I/O

Standard File I/O API (p4.c)

```c
#include <stdio.h>
// report the current location of file pointer in file
long ftell(FILE *fp);

// move file pointer to required location in file
// whence values:
//      SEEK_SET: start location in file
//      SEEK_CUR: current location in file
//      SEEK_END: end location in file
// offset is relative to the whence
int fseek(FILE *fp, long offset, int whence);

// close file
int fclose(FILE *fp);
```

Important

Use man pages to see all the options for each API

Processes

Terminal (p4.c)

The program demonstrates (1) how file descriptor is shared among processes (2) create and read files (3) reading files word by word and line by line

```
# filedes sharing
$ make lab3_p4_1
# word by word reading
$ make lab3_p4_2
# line by line reading
$ make lab3_p4_3
```

Programming Exercise

Activity

1. Create a process chain, where length of the chain (inclusive of root process) is a command line argument (chain1.c)
   - Each process creates a child and waits for the child's execution to complete
   - Each process should print its process id and its parent's process id using execl or execv
   - execl or execv should execute the *echo* command to print the required results (path: /bin/echo)
     echo usage in terminal

```
$ echo Hello
Hello
```

Programming Exercise

Activity

2. Create a process chain, where length of the chain (inclusive of root process) and the output filename (*chain2_out.txt*) are command line arguments (chain2.c)
   - Each process creates a child and waits for the child's execution to complete
   - Each process should write its process id and its parent process id to a file *chain2_out.txt* using Standard file I/O

Programming Exercise

Expected output

Print in bottom up manner with the last child process printing first and the root process printing last

```
$ ./main 3
238542  --  238541
238541  --  238540
238540  --  238532
```

238542 is the last child and 238540 is the root process. The process ids will be different in your output.
Both chain1 and chain2 will have similar outputs with different process ids.

Deliverables

Individual submission: Zip and submit to Gradescope by Sept 26, 11:59pm

1. chain1.c
2. chain2.c
3. chain1_out.txt (copy output of chain1 to the file)
4. chain2_out.txt (generated from chain2)