Lab 2
Processes

CSCI 4061
Introduction to Operating Systems

Sept 18, 2023

UNIVERSITY OF MINNESOTA

Overview

- Command line arguments
- Processes
  - fork
- gdb
- Exercise

Command line arguments

Pass arguments to Executable Object file

```
int main(int argc, char *argv[])
```

- `argc`: Number of command line arguments
- `argv`: Array of command line arguments (array of array of characters)

Command line arguments

Sample code in sample folder(p1.c)

```c
int main(int argc, char *argv[]){
    if(argc != 3){
        printf("Incorrect argument list...\n");
        exit(-1);
    }
    char *pgm = argv[0];
    char *arg1 = argv[1];
    char *arg2 = argv[2];

    print("[%s] has two arguments: [%s] and [%s]\n",
                                    pgm, arg1, arg2);

    return 0;
}
```

Check the Makefile

Terminal

```
$ make lab2_p1_1
$ make clean & make lab2_p1_2 # Error
```

Processes

Process: An instance of a program with individual memory space

fork()

- Creates a new process (child) whose parent is the calling process
- Child receives a copy of the parent address space: **No memory sharing**

Syntax

```
#include <unistd.h>
// returns 0 to child
// returns child process id to parent
// return -1 on failure
pid_t fork(void);
pid_t getpid(void); // returns current process id
pid_t getppid(void); // return parent process id
```

Processes

Sample code in sample folder(p2.c)

```
int main(int argc, char *argv[]){
    int x = 1;
    fork();
    printf("Value of x in process [%d] = [%d]\n",
                                getpid(), x);

    // address of x
    printf("Address of x in process [%d] = [%p]\n",
                                getpid(), &x);
    return 0;
}
```

Terminal

```
$ make lab2_p2
Value of x in process [36267] = [1]
Address of x in process [36267] = [0x7ffe64df0c24]
Value of x in process [36268] = [1]
Address of x in process [36268] = [0x7ffe64df0c24]
```

Processes

Terminal (p2.c)

```
$ make lab2_p2
Value of x in process [36267] = [1]
Address of x in process [36267] = [0x7ffe64df0c24]
Value of x in process [36268] = [1]
Address of x in process [36268] = [0x7ffe64df0c24]
```

Why is the address of x same across processes eventhough the child has a clone of parent's memory space?

Processes

Terminal (p2.c)

```
$ make lab2_p2
Value of x in process [36267] = [1]
Address of x in process [36267] = [0x7ffe64df0c24]
Value of x in process [36268] = [1]
Address of x in process [36268] = [0x7ffe64df0c24]
```

Why is the address of x same across processes even-though the child has a clone of parent's memory space?
Its virtual memory address. You will learn more about it during Memory Management lecture. For now, don't worry about it!!!

Processes

Sample code in sample folder(p3.c)

```c
int main(int argc, char *argv[]){
    int x = 1;
    pid_t pid = fork();
    if(pid == 0){ // Child
        x++;
        printf("Value of x in child [%d] with \
                            parent [%d] = [%d]\n",
                            getpid(), getppid(), x);
        printf("Parent of child process [%d] = %d\n",
                            getpid(), getppid());
    } else {
        x += 3;
        printf("Value of x in parent process [%d] = [%d]\n",
                            getpid(), x);
    }
    return 0;
}
```

Processes

Terminal (p3.c)

```
$ 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 (p3.c)

```
$ 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?

Processes

Terminal (p3.c)

```
$ 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?

When parent process exits before child, a system process (init/systemd with pid = 1 traditionally) becomes the parent of the child

Processes

Terminal (p3.c)

```
$ 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?

When parent process exits before child, a system process (init/systemd with pid = 1 traditionally) becomes the parent of the child

How to ensure parent waits for child?

$\Longrightarrow$ wait() (Next lab)

gdb

GNU Debugger

- Inspect the code execution at certain points
- SEGV fault source location easier to identify
- Use $-g$ flag with *gcc* to create debuggable object file

Debug supported object file

```
$ gcc -g -o main main.c
```

gdb

gdb shell

- Start gdb shell in one of two ways
  1. Call with object file name, say *main*

     ```
     $ gdb ./main
     (gdb)
     ```

  2. Just call gdb and then load the executable using *file*

     ```
     $ gdb
     (gdb) file ./main
     ```

- Once the gdb shell is loaded with the executable, run it using *run* or *r* and required arguments for the program

  ```
  $ gdb
  (gdb) file ./main
  (gdb) r arg1 arg2
  ```

gdb

Sample code in sample folder (p4.c)

The below program will segfault if executed, at line 6. However, the error will not report the line number and it is not caught during compile time.

```c
1   int main(int argc, char *argv[]){
2       int x = 1 + 1;
3       printf("Value of x: %d", x);
4
5       int *arr = NULL;
6       printf("Value of arr: %c", arr[0]); // SEGV
7
8       return 0;
9   }
```

Terminal

```
$ make lab2_p4
./main
make: *** [Makefile:24: lab2_p4] Segmentation fault
                                     (core dumped)
```

gdb

Catch the error using gdb

- Compile code using $-g$ flag
- Run the executable in gdb

Terminal

```
$ gcc -g -o main p4.c
$ gdb ./main
(gdb) r
Program received signal SIGSEGV, Segmentation fault.
0x0000555555555188 in main (argc=1, argv=0x7fffffffdf18)
                                        at p4.c:8
8            printf("Value of arr: %c", arr[0]);
```

gdb

Explore gdb commands

- breakpoints
- watch
- print
- backtrace
- continue
- step

Helpful resource: GDB tutorial

Programming Exercise

Activity

Create *n* child processes for a single parent process (fan.c)

1. The value of *n* is passed to the executable via command line
2. If the number of arguments passed is incorrect, error and exit the code
3. Use atoi() to convert input character array to integer *n*
4. Each process should print its process id and its parent id

Programming Exercise

Expected output

```
$ ./main 3
Child [24292] --- Parent [24291]
Child [24291] --- Parent [24277]
Child [24293] --- Parent [24291]
Child [24294] --- Parent [24291]
```

24291 is the parent, 24292, 24293 and 24294 are the children.
24277 is the parent of the parent process.
The output order and process ids will be different at your end.
Please ensure to test with a value of n $\leq$ 3 as improper spawning of processes may lead to your system freezing.
Copy the result on the screen to a new file output.txt

Deliverables

Individual submission: Zip and submit to Gradescope by Sept 19, 11:59pm

1. fan.c
2. output.txt