Lab: Process API

Outcomes

- Program Execution and context.
- Create and Control Processes
- fork(), wait(), and exec() signal calls

Online Resources

- **Operating Systems: Three Easy Pieces:** https://pages.cs.wisc.edu/~remzi/OSTEP/
- **Homework page:** https://pages.cs.wisc.edu/~remzi/OSTEP/Homework/homework.html
- **Process API Reading**: https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf

---

Questions

1. Program Execution and Hardware Elements

- When your program runs, which hardware elements are used?
  Answer:

2. Context During Execution

- Define "context" in terms of program execution. List elements of the context you observed in the online emulator.
  Answer:

3. Switching Programs and Context Handling

- If you pause execution of a program to run another program, how can you return to the previous program?
  Answer:

- How does the OS handle simultaneous processes that use the same registers?
  Answer:

- Compare the states you observed in `top` to the states you expected from reading the lecture slides. Are there any differences?
  Answer:

- Why is stopping a program important? Use `top` outputs to explain.
  Answer:

Task 1

Read about Process API and run the code examples as you go.

I have pasted the code segments below so it is easier for you to copy and save.

p1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    printf("hello (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {
        // fork failed
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        // child (new process)
        printf("child (pid:%d)\n", (int) getpid());
    } else {
        // parent goes down this path (main)
        printf("parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
        return 0;
}
```

How to Compile and Run:

1. Save the code as p1.c.
2. Compile:

   ```
   gcc -g -o p1 p1.c
   ```

3. Run:

   ```
   ./p1
   ```

4. When there is a prompt in the reading, run the command on your machine.
   For Example:

   ```
   prompt> ./p1
    .....
   prompt> cat p4.output
   ```

Understanding the Code

- Open p1.c (and p2.c , p3.c , p4.c) in a text editor (e.g., pico).

  ```
  pico p1.c
  ```

- Edit the code and explore.
- Take a screenshot of the output of the prompts from the reading.
- Summarize (briefly) what is happening in each of p1, p2, p3 and p4

  Answer:

p2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    printf("hello (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) { // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("child (pid:%d)\n", (int) getpid());
    } else { // parent goes down this path
        int rc_wait = wait(NULL);
        printf("parent of %d (rc_wait:%d) (pid:%d)\n",
        rc, rc_wait, (int) getpid());
    }
        return 0;
}
```

p3.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    printf("hello (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc");
        // program: "wc"
        myargs[1] = strdup("p3.c"); // arg: input file
        myargs[2] = NULL;
        // mark end of array
        execvp(myargs[0], myargs); // runs word count
        printf("this shouldn’t print out");
    } else {
        // parent goes down this path
        int rc_wait = wait(NULL);
        printf("parent of %d (rc_wait:%d) (pid:%d)\n",
        rc, rc_wait, (int) getpid());
    }
    return 0;
}
```

p4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    int rc = fork();
    if (rc < 0) {
        // fork failed
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        // child: redirect standard output to a file
        close(STDOUT_FILENO);
        open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC,S_IRWXU);
        // now exec "wc"...
        char *myargs[3];
        myargs[0] = strdup("wc"); // program: wc
        myargs[1] = strdup("p4.c"); // arg: file to count
        myargs[2] = NULL; // mark end of array
        execvp(myargs[0], myargs); // runs word count
    } else {
        // parent goes down this path (main)
        int rc_wait = wait(NULL);
    }
    return 0;
}
```