CSC 112: Computer Operating Systems

Review Questions

Department of Computer Science,
Hofstra University

Q. Fork

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative.
- 1. Will the parent and child print the same value for a?
- ANS: Yes. Processes do not share the same memory space, so a is 2 for both.
- 2. Will they print the same memory address &a?
- ANS: Yes. Fork copies the address space of the parent to the child.
- 3. Will they write to the same STDOUT?
- ANS: Yes. File descriptors are copied over to the new process, so both STDOUTs will reference the same ”file”.

```
1 int main(void) {
2 int a = 1;
3 pid_t fork_ret = fork();
4 if (fork_ret > 0) {
5   a++;
6   fprintf(stdout, "Parent: int a is %d at %p\n",
a, &a);
7 } else if (fork_ret == 0) {
8   a++;
9   fprintf(stdout, "Child: int a is %d at %p\n", a,
&a);
10 } else {
11   printf("Oedipus");
12 }
13 }
```

Q. Fork

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative.
- 1. What does this program print?
- ANS: Currently, the program stops after printing 3, giving an output of
  - 0
  - 1
  - 2
  - 3
  - <output of ls>

```
1 int main(void) {
2 char** argv = (char**) malloc(3 * sizeof(char*));
3 argv[0] = "/bin/ls";
4 argv[1] = ".";
5 argv[2] = NULL;
6 for (int i = 0; i < 10; i++) {
7   printf("%d\n", i);
8   if (i == 3) {
9       execv("/bin/ls", argv);
10  }
11 }
12 return 0;
13 }
```

Q. Fork

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative.
- 1. What does this program print?
- ANS: The parent process will print from 0 to 9. The child process will print <output of ls> with any possible interleaving with the parent's printing of 0 to 9.
  - 0
  - 1
  - 2
  - …
  - 9
  - <output of ls>

```
1 int main(void) {
2 char** argv = (char**) malloc(3 * sizeof(char*));
3 argv[0] = "/bin/ls";
4 argv[1] = ".";
5 argv[2] = NULL;
6 for (int i = 0; i < 10; i++) {
7  printf("%d\n", i);
8  if (i == 3) {
9    pid_t fork_ret = fork();
10   if (fork_ret == 0)
11     execv("/bin/ls", argv);
12 }
12 return 0;
13 }
```

Q. Dining Lawyers

- Consider the Dining Lawyers problem. There are 3 lawyers P1 to P3, each with a different number of arms. P1 has 1 arm and needs 1 fork to eat; P2 has 2 arms and needs 2 forks to eat; P3 has 3 arms and needs 3 forks to eat. There is a pile of 3 forks at center of the table. Each lawyer picks up one fork at a time, and when he gets enough forks, he eats and then puts down all his forks.
- Is it possible for the system to be deadlocked? If no, explain why. If yes, show a (potential) deadlock state and run Banker's algorithm to check it. (You need to give the Max, Allocation, Need matrices, Total and Available vectors, and Available resources after completion of each process.)

Q. Dining Lawyers Solution: 3 Lawyers, each with 1,2,3 arms, 3 forks

Initially, all forks are free.

Max	Allocation
1	0
2	0
3	0

Total	Available
3	3

P2 grabs 1 fork and P3 grabs 2 forks

Max	Allocation	Need
1	0	1
2	1	1
3	2	1

Total	Available
3	0

Current state is a deadlock.

Available resources after completion of each process

	R1
Init	0
Deadlock	