Multiprocessing: fork & wait; exec;

COSC 208, Introduction to Computer Systems, 2022-04-21

Outline

- Process abstraction (review)
- Creating processes
- Waiting for processes
- Running a different program `exec`

Waiting for processes

- Wait for any child to finish — `int wait(int *status)`
    - Returns PID of the child process that finished
    - `status` parameter is optional
        - if passed a valid integer pointer, `wait` will store the return value of the child process's `main` function at the referenced memory location
        - if return value of child process's main function is not needed, then pass `NULL`
- Wait for a specific process to finish — `int waitpid(pid_t pid, int *status, int options)`
    - Returns PID of the process that finished
    - `pid` is PID if process to wait for — need not be a child process
    - `status` is the same as `wait`
    - `options` is typically `0`, except in special circumstances
- Wait functions do not return until child or specific process, respectively, finishes
- Q1: *What are all possible outputs of this program?*

```c
int main() {
    int pid = fork();
    if (pid == 0) {
        printf("Child\n");
    } else {
        wait(NULL);
        printf("Parent\n");
    }
    return 0;
}
```

```
Child
Parent
```

Q2: *What are all possible outputs of this program (assuming the new process has PID 13346)?*

```C
int main() {
    int pid = fork();
    printf("A %d\n", pid);
    if (pid == 0) {
        printf("B\n");
    } else {
        wait(NULL);
        printf("C\n");
    }
}
```

```
A 0
A 13346
B
C
```

OR

```
A 13346
A 0
B
C
```

OR

```
A 0
B
A 13346
C
```

Running a different program

- `int exec(const char *path, const char *argv[])`
  - Used to switch which program is running in a process — replaces current code with code for a new program and starts executing that program from main
  - `path` == full path to program
  - `argv` == array of strings containing the full path to the program, any command line arguments, and `NULL`
- Example program

```c
int main(int argc, char **argv) {
    printf("Begin\n");
    int pid = fork();
    if (pid == 0) {
        printf("Child\n");
        char *cmd[] = { "/usr/bin/date", NULL };
        execv(cmd[0], cmd);
    } else {
        printf("Parent\n");
    }
    printf("End\n");
    return 0;
}
```

```
Begin
Parent
End
Child
Mov Nov 8 13:20:00 UTC 2021
```

- Q3: *What is the output produced by running `./progA`, assuming no errors occur?* **progA:**

```c
int main() {
    pid_t a = fork();
    if (a == 0) {
        execv("./progB", NULL);
        printf("A 2nd gen\n");
        return 0;
    } else {
        wait(NULL);
        printf("A 1st gen\n");
        return 0;
    }
}
```

progB:

```c
int main() {
    pid_t b = fork();
    if (b == 0) {
        printf("B 2nd gen\n");
        return 0;
    } else {
        wait(NULL);
        printf("B 1st gen\n");
        return 0;
    }
}
```

```
B 2nd gen
B 1st gen
A 1st gen
```

Practice

- QA: *What does the following code output?*

```
int main(int argc, char **argv) {
    int value = 100;
    int pid = fork();
    if (pid == 0) {
        value -= 50;
    } else {
        value += 50;
    }
    printf("My value is %d\n", value);
    return 0;
}
```

```
My value is 50
My value is 150
```

OR

```
My value is 150
My value is 50
```

- QB: *What does the following code output?*

```
int main(int argc, char **argv) {
    printf("Begin\n");
    int pid = fork();
    if (pid == 0) {
        printf("Child\n");
        return 0;
    } else {
        printf("Parent\n");
    }
    printf("End\n");
}
```

```
Begin
Child
Parent
End
```

OR

```
Begin
Parent
Child
End
```

OR

```
Begin
Parent
End
Child
```

- QC: *How would you modify the above program such that `Child` always prints before `Parent`?*

```c
int main(int argc, char **argv) {
    printf("Begin\n");
    int pid = fork();
    if (pid == 0) {
        printf("Child\n");
        return 0;
    } else {
        wait(NULL);
        printf("Parent\n");
    }
    printf("End\n");
}
```