submission-2.md

OS Lab Week-2

- Name : P K Navin Shrinivas
- Section : D
- SRN : PES2UG20CS237

Program 1 : fork(), pid(), ppid()

In this program we first use fork for the first time, this is part of the unistd.h library fork takes no parameters but returns conditionally, what do I mean by conditionally?

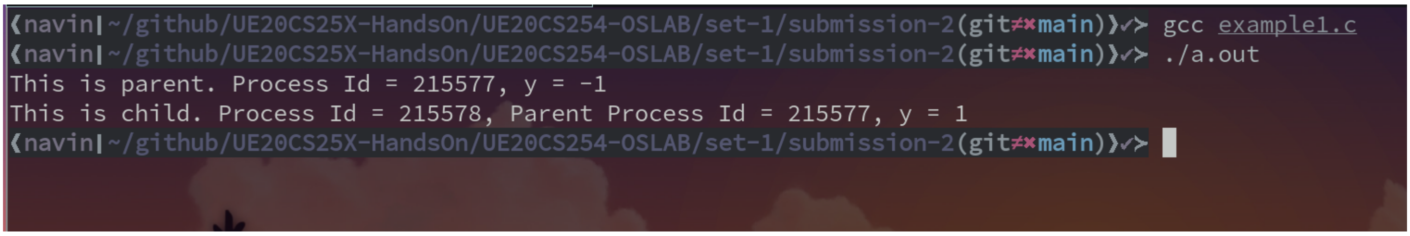
Negative Value: creation of a child process was unsuccessful. Zero: Returned to the newly created child process. Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

more important things of fork function :

Important: Parent process and child process are running the same program, but it does not mean they are identical. OS allocate different data and states for these two processes, and the control flow of these processes can be different.

getpid() gets the process ID of the current process getppid() gets the process ID of the parent proces

Screenshot :

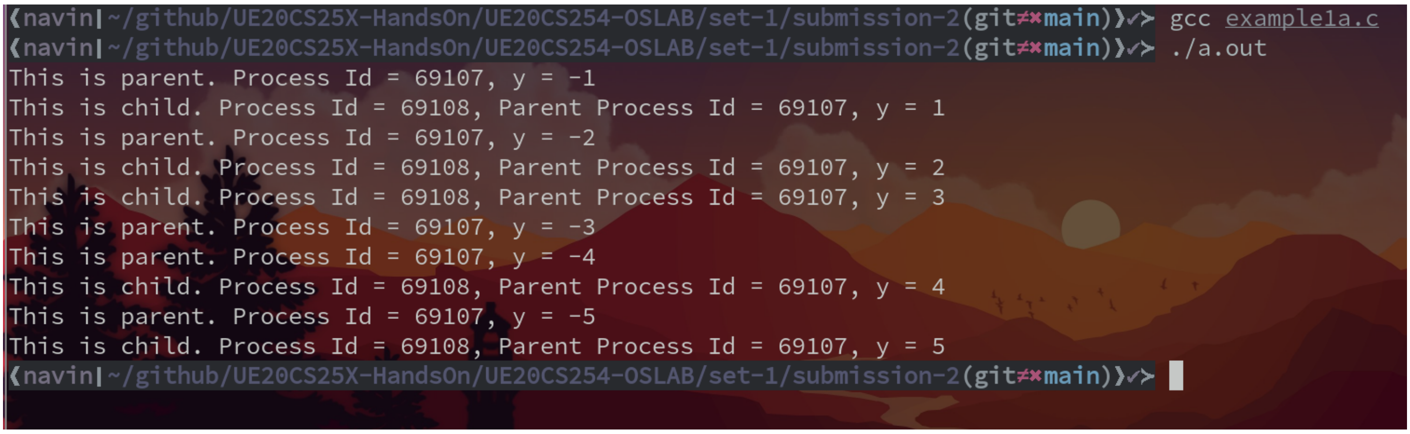


```
❮navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❯✓> gcc example1.c
❮navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❯✓> ./a.out
This is parent. Process Id = 215577, y = -1
This is child. Process Id = 215578, Parent Process Id = 215577, y = 1
❮navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❯✓> █
```

Program 1a :

we use the same functions as in program 1, no new functions. But this program better represents how OS allocates the parents and child their seperate data and states.

Screenshot :



```
❮navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❯✓> gcc example1a.c
❮navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❯✓> ./a.out
This is parent. Process Id = 69107, y = -1
This is child. Process Id = 69108, Parent Process Id = 69107, y = 1
This is parent. Process Id = 69107, y = -2
This is child. Process Id = 69108, Parent Process Id = 69107, y = 2
This is child. Process Id = 69108, Parent Process Id = 69107, y = 3
This is parent. Process Id = 69107, y = -3
This is parent. Process Id = 69107, y = -4
This is child. Process Id = 69108, Parent Process Id = 69107, y = 4
This is parent. Process Id = 69107, y = -5
This is child. Process Id = 69108, Parent Process Id = 69107, y = 5
❮navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❯✓> █
```

Program 2 : wait(NULL)

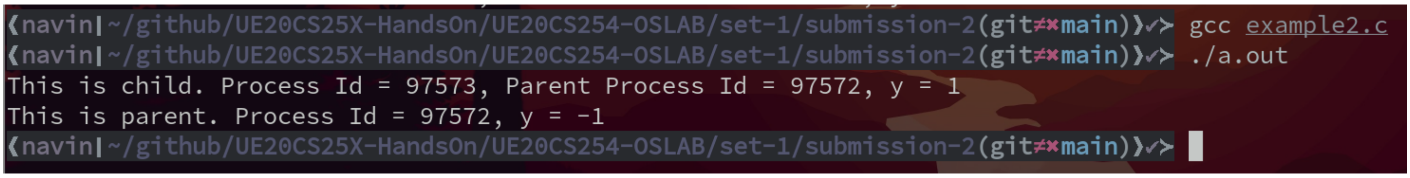
wait(NULL) makes the parent wait for the exit status of child, now 2 cases are possible :

Child finishes before parent reaches wait(NULL): In such a case the child becomes a zombie proccess and waits for the parent process to read its exit status.

Child continues afte parent reached wait(NULL) : In this case the parent waits for the child to return back a exit status.

wait(NULL) provides us programmer with a sure shot way of avoiding orphan process.

Screenshot :

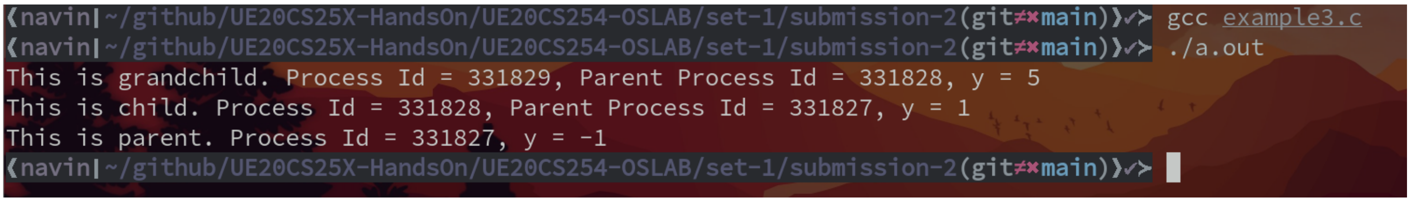


```
⟨navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)⟩✓> gcc example2.c
⟨navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)⟩✓> ./a.out
This is child. Process Id = 97573, Parent Process Id = 97572, y = 1
This is parent. Process Id = 97572, y = -1
⟨navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)⟩✓>
```

Program 3 :

We use same functions as last program, no new functions. But here we make a grandchild, Figuring out the order of outputs [due to wait(NULL)] along with number output proved to be challenging.

Screenshots :



```
⟨navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)⟩✓> gcc example3.c
⟨navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)⟩✓> ./a.out
This is grandchild. Process Id = 331829, Parent Process Id = 331828, y = 5
This is child. Process Id = 331828, Parent Process Id = 331827, y = 1
This is parent. Process Id = 331827, y = -1
⟨navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)⟩✓>
```

Program 4 : execl() execv()

Both the functions are used to execute a file, In linux all commands are simply binary files located in /bin/ ans often part of path.

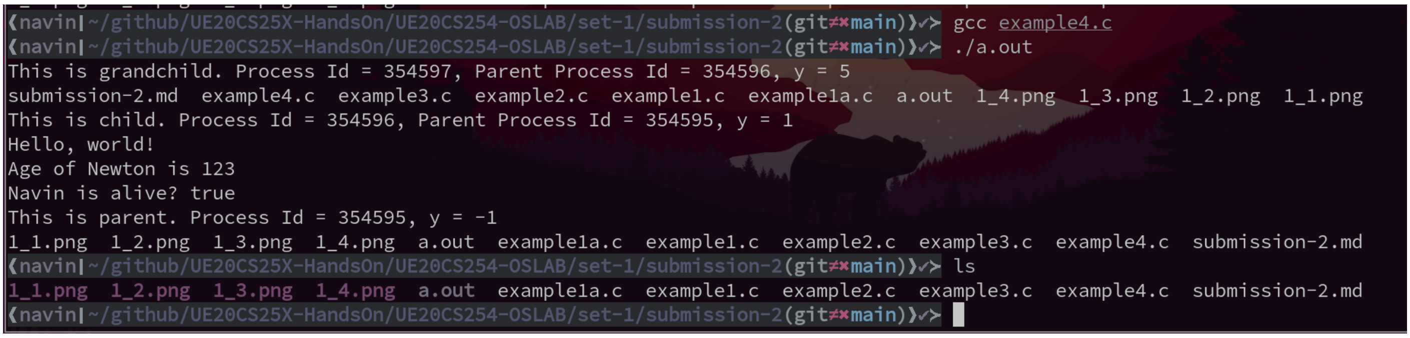
execX() functions are usefull to execute these files.The manpages gives a very clear understanding of these two functions and their uses :

execl : Used to execute a file where first argument is the path to the file, all other parameters are arguments that need to to be passed to the file. These aguments need to be terminated with NULL as per posix standards so that execl can know the end of arguments, which is seen in code example with the last parameter being NULL.

execv : Used to execute files where the arguments are passed in a vector [in C, arrays], and just like before the vector must be terminated with a NULL which is also seen in code when the only thing in array "a" is NULL, indicating no arguments.

also the -r parameter to execl is a flag being passed as argument to ls which tells ls to sort output in reverse order

Screenshot :



```
❬navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❭✓➤ gcc example4.c
❬navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❭✓➤ ./a.out
This is grandchild. Process Id = 354597, Parent Process Id = 354596, y = 5
submission-2.md  example4.c  example3.c  example2.c  example1.c  example1a.c  a.out  1_4.png  1_3.png  1_2.png  1_1.png
This is child. Process Id = 354596, Parent Process Id = 354595, y = 1
Hello, world!
Age of Newton is 123
Navin is alive? true
This is parent. Process Id = 354595, y = -1
1_1.png  1_2.png  1_3.png  1_4.png  a.out  example1a.c  example1.c  example2.c  example3.c  example4.c  submission-2.md
❬navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❭✓➤ ls
1_1.png  1_2.png  1_3.png  1_4.png  a.out  example1a.c  example1.c  example2.c  example3.c  example4.c  submission-2.md
❬navin|~/github/UE20CS25X-HandsOn/UE20CS254-OSLAB/set-1/submission-2(git≠×main)❭✓➤
```