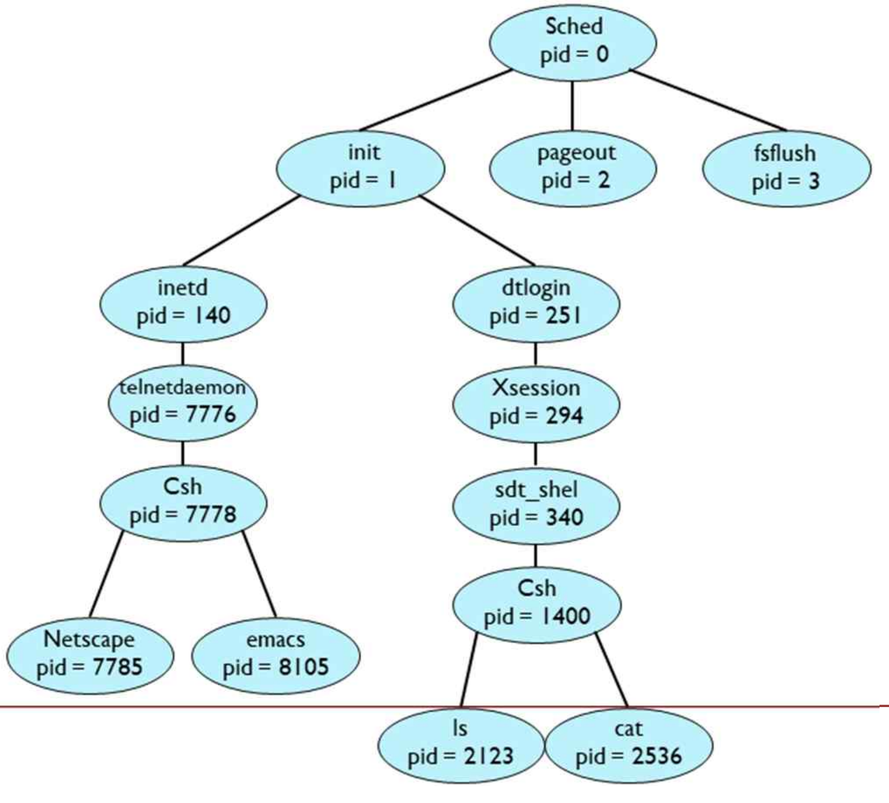
Operating System: Process API

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

Process Creation

- Process creation
  - Parent process creates child processes,
  - which, in turn creates other processes
  - Finally, it forms a tree of processes



Process Creation

- Child processes need resources
  - OS gives, or
  - Parent shares

운영체제가 직접주거나 부모와 나누는 (전체 혹은 일부)

- Resource sharing
  - Parent and children share all resources,
  - Children share subset of parent’s resources, or
  - Parent and child share no resources

& 기호를 통한 백그라운드 실행의 경우

- Execution
  - Parent and child execute concurrently, or
  - Parent waits until child terminate
- Address space
  - Child duplicates parent, or
  - Child has a new program loaded into it

Process Creation

- Summary of process creation
  - Create PCB within OS kernel
  - Allocate memory space
  - Load binary program
  - Initialize the program
- UNIX example
  - fork system call creates new process
    - ➢ duplicate parent’s PCB
    - ➢ allocate memory space
  - execve system call is used after a fork
    - ➢ load binary program from disk
    - ➢ initialization

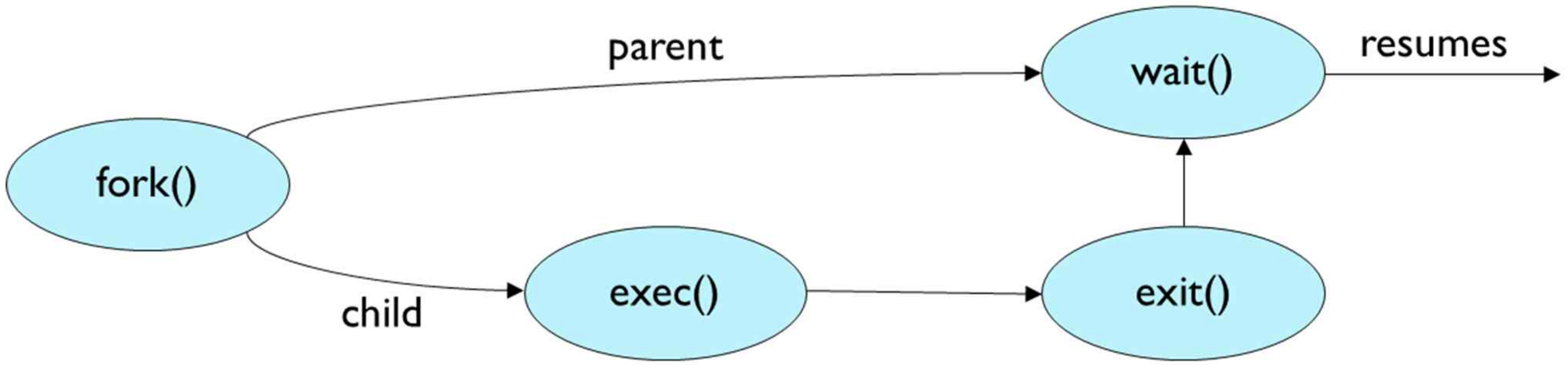
Process Termination

- Process executes last statement and asks operating system to delete itself (**exit**) 정상종료
  - Child process returns a status value to its parent (wait)
  - Child process' resources are deallocated by operating system

- Parent may terminate execution of child processes (abort) 비정상종료.
  - If child has exceeded the allocated resources
  - If task assigned to child is no longer required

Process Creation & Termination

- fork, exec, exit, and wait system call



The fork() System Call

- Create a new process
  - The newly-created process has its own copy of the address space, registers, and PC

p1.c

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

int main(int argc, char *argv[]){
    printf("hello world (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("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

Calling fork() example (Cont.)

Result (Not deterministic)

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

or

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

The wait() System Call

- This system call won't return until the child has run and exited

p2.c

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

int main(int argc, char *argv[]){
    printf("hello world (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("hello, I am child (pid:%d)\n", (int) getpid());
    } else {                // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
        rc, wc, (int) getpid());
    }
    return 0;
}
```

The wait() System Call (Cont.)

Result (Deterministic)

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

The exec() System Call

- Run a program that is different from the calling program

p3.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 world (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("hello, I am child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc");         // program: "wc" (word count)
        myargs[1] = strdup("p3.c");       // argument: file to count
        myargs[2] = NULL;                 // marks end of array
        …
```

ls . cd 와같은 명령어.

The exec() System Call (Cont.)

p3.c (Cont.)

```
…
        execvp(myargs[0], myargs); // runs word count
        printf("this shouldn't print out");
    } else {                        // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

Result

```
prompt> ./p3
hello world (pid:29383)
hello, I am child (pid:29384)
29 107 1030 p3.c
hello, I am parent of 29384 (wc:29384) (pid:29383)
prompt>
```

p3.c 파일의 line, word, bytes를 출력.

All of the above with redirection

p4.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; exit
        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);
        …
```

All of the above with redirection (Cont.)

p4.c

```
        …
        // now exec "wc"...
        char *myargs[3];
        myargs[0] = strdup("wc");           // program: "wc" (word count)
        myargs[1] = strdup("p4.c");         // argument: file to count
        myargs[2] = NULL;                   // marks end of array
        execvp(myargs[0], myargs);          // runs word count
    } else {                            // parent goes down this path (main)
        int wc = wait(NULL);
    }
    return 0;
}
```

Result

```
prompt> ./p4
prompt> cat p4.output
32 109 846 p4.c
prompt>
```