

# Process (1A)

---

- Process Command

Copyright (c) 2012 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using OpenOffice and Octave.

# Child Process

```
young@USys7:~$ vi s1
young@USys7:~$
young@USys7:~$
young@USys7:~$ chmod +x s1
young@USys7:~$ ./s1
PID = 38508
Parents' PID = 38421
variable i =
young@USys7:~$ i= 100
100: command not found
young@USys7:~$ i=100
young@USys7:~$ ./s1
PID = 38520
Parents' PID = 38421
variable i =
young@USys7:~$ echo $$
38421
young@USys7:~$ export i
young@USys7:~$ ./s1
PID = 38543
Parents' PID = 38421
variable i = 100
young@USys7:~$
```

```
young@USys7:~$ cat s1
#!/bin/bash

echo "PID = $$"
echo "Parents' PID = $PPID"
echo "variable i = $i"
```

# Child Process

```
young@USys7:~$ cat s1
#!/bin/bash

echo "s1's PID = $$"
echo "Parents' PID = $PPID"
echo "variable i = $i"

young@USys7:~$ cat s2
#!/bin/bash

i=222

export i

echo "s2's PID = $$"
echo "variable i = $i"

echo "call ./s1"
./s1
```

```
young@USys7:~$ ./s1
s1's PID = 39364
Parents' PID = 38421
variable i = 100
young@USys7:~$ ./s2
s2's PID = 39365
variable i = 222
call ./s1
s1's PID = 39366
Parents' PID = 39365
variable i = 222
```

# Child Process

```
young@USys7:~$ cat m1.c
```

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

```
int main() {
    pid_t pid = getpid();          // Get current process ID
    pid_t ppid = getppid();        // Get parent process ID
    char *i = getenv("i");         // Get environment variable "i"

    printf("m1's PID = %d\n", (int) pid);
    printf("Parents' PID = %d\n", (int) ppid);
    printf("variable i = %d\n", atoi(i));
}
```

```
young@USys7:~$ ./m2
```

```
m2's PID = 40240
variable i = 222
call ./m1
m1's PID = 40242
Parents' PID = 40241
variable i = 222
execl: Success
```

```
young@USys7:~$ cat m2.c
```

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

```
int main() {
    // Set environment variable i = 222
    setenv("i", "222", 1); // 1 = overwrite if already set

    // Print current process ID
    pid_t pid = getpid();
    printf("m2's PID = %d\n", pid);

    // Print value of i
    char *i = getenv("i");
    printf("variable i = %d\n", atoi(i));

    // Print message and call ./s1
    printf("call ./m1\n");
    system("./m1");

    // If execl fails
    perror("execl");
    return 1;
}
```

# Child Process

```
young@USys7:~$ cat m1.c
```

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

```
int main() {
    pid_t pid = getpid();          // Get current process ID
    pid_t ppid = getppid();        // Get parent process ID
    char *i = getenv("i");         // Get environment variable "i"

    printf("m1's PID = %d\n", (int) pid);
    printf("Parents' PID = %d\n", (int) ppid);
    printf("variable i = %d\n", atoi(i));
}
```

```
young@USys7:~$ ./m2
```

```
m2's PID = 40360
variable i = 222
call ./m1
m1's PID = 40360
Parents' PID = 38421
variable i = 222
```

```
young@USys7:~$ cat m2.c
```

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

```
int main() {
    // Set environment variable i = 222
    setenv("i", "222", 1); // 1 = overwrite if already set

    // Print current process ID
    pid_t pid = getpid();
    printf("m2's PID = %d\n", pid);

    // Print value of i
    char *i = getenv("i");
    printf("variable i = %d\n", atoi(i));

    // Print message and call ./s1
    printf("call ./m1\n");
    execl("./m1", "./m1", (char *)NULL);

    // If execl fails
    perror("execl");
    return 1;
}
```

# fork()

```
// f1.c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(void) {
    printf("* Before fork :");
    printf(" pid = %d", getpid());
    printf(" ppid = %d \n", getppid());

    fork();

    printf("* After fork :");
    printf(" pid = %d", getpid());
    printf(" ppid = %d \n", getppid());
}

$ gcc -Wall -o f1 f1.c
$ ./f1
* Before fork : pid = 4871 ppid = 3919
* After fork : pid = 4871 ppid = 3919
* After fork : pid = 4872 ppid = 4871
```

```
// f2.c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(void) {
    int rval;

    printf("* Before fork :");
    printf(" pid = %d", getpid());
    printf(" ppid = %d \n", getppid());

    rval = fork();

    printf("* After fork :");
    printf(" pid = %d", getpid());
    printf(" ppid = %d ", getppid());
    printf(" rval = %d \n", rval);
}

$ gcc -Wall -o f2 f2.c
$ ./f2
* Before fork : pid = 5161 ppid = 3919
* After fork : pid = 5161 ppid = 3919 rval = 5162
* After fork : pid = 5162 ppid = 5161 rval = 0
```

# fork()

```
// f3.c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
```

```
int main(void) {
```

```
    int rval;
```

```
    printf("* Before fork :");
    printf("  pid = %d", getpid());
    printf("  ppid = %d \n", getppid());
```

```
    rval = fork();
```

```
    if (rval == 0) {
```

```
        printf("* After fork : (rval = %d) == 0 \n", rval);
        printf("  pid = %d", getpid());
        printf("  ppid = %d \n", getppid());
```

```
    } else {
```

```
        printf("* After fork : (rval = %d) != 0 \n", rval);
        printf("  pid = %d", getpid());
        printf("  ppid = %d \n", getppid());
```

```
    }
```

```
    printf("* After fork :");
    printf("  pid = %d", getpid());
    printf("  ppid = %d ", getppid());
    printf("  rval = %d \n", rval);
```

```
}
```

```
gcc -Wall -o f3 f3.c
```

```
young@USys7:~/Desktop/SysProg.Lab/Lab9.fork$ ./f3
```

```
* Before fork :  pid = 6098  ppid = 3919
```

```
* After fork : (rval != 0) :  pid = 6098  ppid = 3919  rval = 6099
```

```
* After fork :  pid = 6098  ppid = 3919  rval = 6099
```

```
* After fork : (rval == 0) :  pid = 6099  ppid = 6098  rval = 0
```

```
* After fork :  pid = 6099  ppid = 6098  rval = 0
```

# fork()

```
// f4.c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int rval;
    int status;

    printf("* Before fork :");
    printf(" pid = %d", getpid());
    printf(" ppid = %d \n", getppid());

    rval = fork();

    if (rval == 0) {
        printf("* After fork : (rval = %d) == 0 :", rval);
        printf(" pid = %d", getpid());
        printf(" ppid = %d \n", getppid());
        exit(99);
    } else {
        wait(&status);

        printf("* After fork : (rval = %d) != 0 :", rval);
        printf(" pid = %d", getpid());
        printf(" ppid = %d \n", getppid());
        exit(0);
    }
}
```



```
printf("* After fork :");
printf(" pid = %d", getpid());
printf(" ppid = %d ", getppid());
printf(" rval = %d \n", rval);

}
```

```
$ ./f4
* Before fork : pid = 8221 ppid = 6294
* After fork : (rval = 0) == 0 : pid = 8222 ppid = 8221
* After fork : (rval = 8222) != 0 : pid = 8221 ppid = 6294
```

# Child Process

```
young@young-530U3C-530U4C:~$  
young@young-530U3C-530U4C:~$  
young@young-530U3C-530U4C:~$ ps  
  PID TTY          TIME CMD  
 2208 pts/1        00:00:00 bash  
 2362 pts/1        00:00:00 ps  
young@young-530U3C-530U4C:~$ xterm &  
[1] 2398  
young@young-530U3C-530U4C:~$  
young@young-530U3C-530U4C:~$  
young@young-530U3C-530U4C:~$ ps  
  PID TTY          TIME CMD  
 2208 pts/1        00:00:00 bash  
 2398 pts/1        00:00:00 xterm  
 2454 pts/1        00:00:00 ps  
young@young-530U3C-530U4C:~$ ps  
  PID TTY          TIME CMD  
 2208 pts/1        00:00:00 bash  
 2398 pts/1        00:00:00 xterm  
 2500 pts/1        00:00:00 ps  
young@young-530U3C-530U4C:~$ pstree
```

Job # → [1] 2398

background → xterm &

child process → 2398 pts/1 00:00:00 xterm

# pstree

```
gnome-terminal—bash—pstree
                  |
                  |—xterm—bash—octave
                  |
                  |—gnome-pty-helpe
                  |
                  |—3*[{gnome-terminal}]
```

chmod

chown

chgrp

# ps -ef

ps -ef

| UID   | PID  | PPID | C | STIME | TTY   | TIME     | CMD                              |
|-------|------|------|---|-------|-------|----------|----------------------------------|
| root  | 1    | 0    | 0 | 08:22 | ?     | 00:00:00 | /sbin/init                       |
| root  | 2    | 0    | 0 | 08:22 | ?     | 00:00:00 | [kthreadd]                       |
| root  | 3    | 2    | 0 | 08:22 | ?     | 00:00:00 | [ksoftirqd/0]                    |
| root  | 6    | 2    | 0 | 08:22 | ?     | 00:00:00 | [migration/0]                    |
| root  | 7    | 2    | 0 | 08:22 | ?     | 00:00:00 | [watchdog/0]                     |
| root  | 8    | 2    | 0 | 08:22 | ?     | 00:00:00 | [migration/1]                    |
| young | 2199 | 1    | 0 | 08:28 | ?     | 00:00:07 | gnome-terminal                   |
| young | 2207 | 2199 | 0 | 08:28 | ?     | 00:00:00 | gnome-pty-helper                 |
| young | 2208 | 2199 | 0 | 08:28 | pts/1 | 00:00:00 | bash                             |
| lp    | 2361 | 870  | 0 | 08:28 | ?     | 00:00:00 | /usr/lib/cups/notifier/dbus dbus |
| young | 2398 | 2208 | 0 | 08:29 | pts/1 | 00:00:00 | xterm                            |
| young | 2400 | 2398 | 0 | 08:29 | pts/2 | 00:00:00 | bash                             |
| young | 2497 | 2400 | 0 | 08:29 | pts/2 | 00:00:00 | octave                           |

# Process Related Command

---

```
pstree -h -p -n -l  
pstree -u  
ps -ef
```

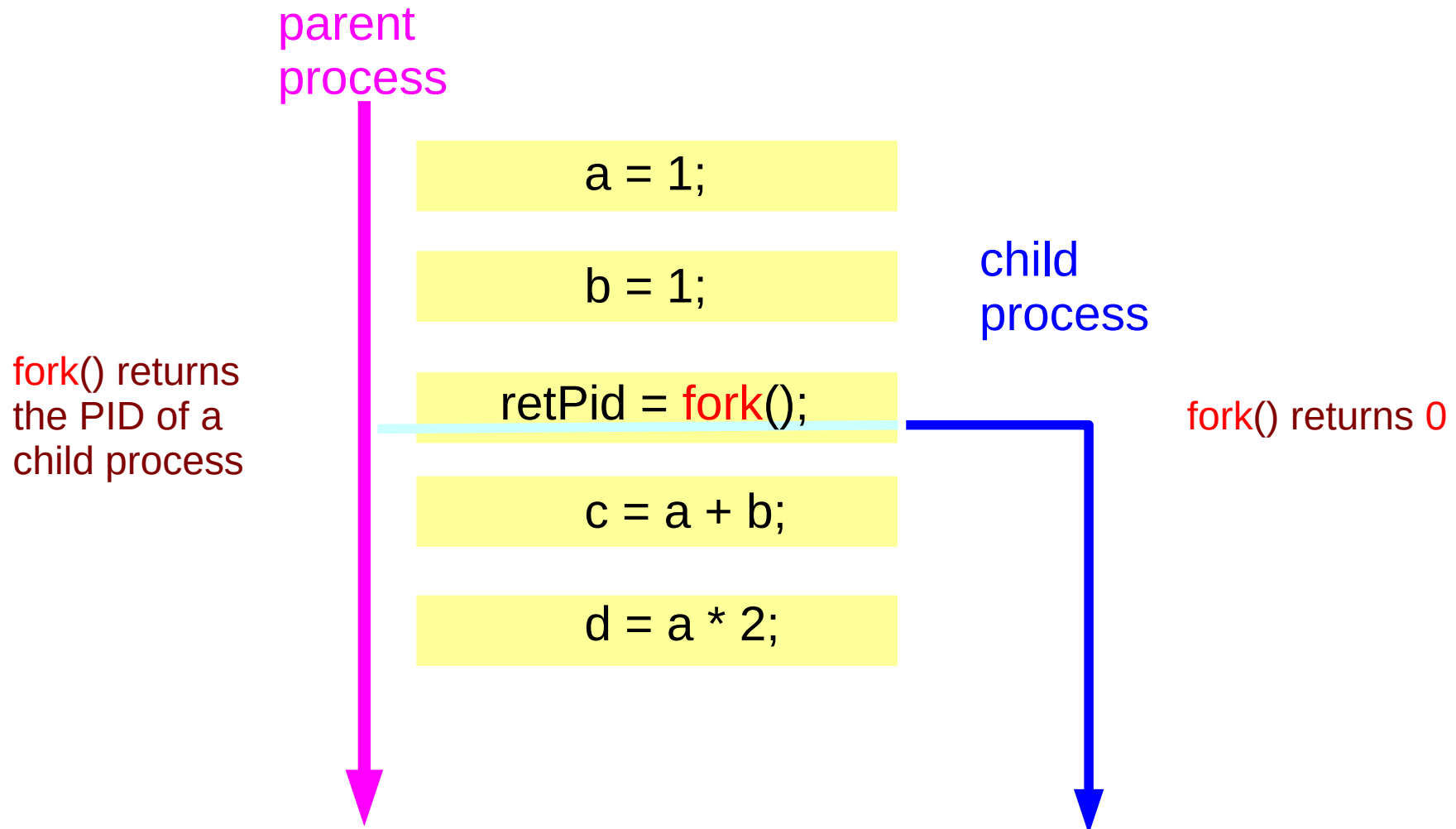
```
bg  
fg  
jobs
```

```
kill -9  
kill -l  
killall
```

```
fuser  
pidof  
top
```

```
ntsysv  
chkconfig
```

# fork()



# fork() example (1)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
    int x, n;
    int retPID = 9999;
    char *message;

    x = 0;
    printf("x = %d pid=%d returned PID = %d\n", x, getpid(), retPID);

    retPID = fork();

    x = 1;
    printf("x = %d pid=%d returned PID = %d\n", x, getpid(), retPID);
```

## fork() example (2)

```
if (retPID == 0) { // child process
    printf("I am a child process %d ParentPID = %d \n", getpid(), getppid());
    n = 5;
    message = "I am a child process";
}
else {
    printf("I am a parent process %d ChildPID = %d \n", getpid(), retPID);
    n = 3;
    message = "I am a parent process";

    exec1("/usr/bin/firefox", "firefox" , (char *) 0);
}

for ( ; n > 0; n--) {
    puts(message);
    sleep(3);
}

return 0;
}
```

# argc, argv example

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char * argv[])
{
    int n=0;

    for (n =0; n < argc; n++) {
        printf("n=%d argv[%d] = %s \n", n, n, argv[n]);
    }

    return 0;
}
```

# SetUID Example

```
young/SysP$ vi fprn.c
young/SysP$ gcc fprn.c -o fprn
young/SysP$ chmod 4755 fprn
young/SysP$ ls -l fprn
```

```
mat$ /home/young/SysP/fprn
```

```
young/SysP$ chmod 755 fprn
young/SysP$ ls -l fprn
```

```
mat$ /home/young/SysP/fprn
```

```
#include <stdio.h>

void main()
{
    FILE *fp;

    // fp = fopen("/home/young/SysP/fprn.out", "w");
    fp = fopen("fprn.out", "w");

    if (fp != NULL) {
        printf("Hello, world!\n");
        fprintf(fp, "Hello, world!\n");
        fclose(fp);
    }
    else {
        error("Cannot open /home/young/SysP/fprn.out \n");
    }
}
```

# Exit Code

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

#include <unistd.h>

int main(int argc, char* argv[])
{
    int rpid, code, exitno;

    if (fork() == 0) { // child process
        execl("/home/young/SysP/ret34", "ret34", (char *) 0);
    }
    else { // parent process
        wait(&code);
        exitno = WEXITSTATUS(code);
        printf("exitno = %d %x \n", exitno, exitno);
        printf("return code = %d WEXITSTATUS(%x)=%x\n",
            code, code, WEXITSTATUS(code));
    }
}
```

## ret34

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

int main(int argc, char * argv[])
{
    return(34); // should be one byte (0~255)
    // exit(34);
}
```

## In bash

```
young:~/SysP$ ./ret34
young:~/SysP$ echo $?
34
young:~/SysP$
```

# Sticky Bit

---

```
bob$ cd /home/mat/SharedDir
bob$ vi bob.file
bob$ ls -al .

bob$ rm mat.file
```

```
mat$ cd SharedDir
mat$ vi mat.file
mat$ ls -al .

mat$ rm bob.file
```

# Bad Fork Example (1)

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

int main(int argc, char* argv[])
{
    int i, rpid, code, exitno;

    for (i = 0; i < 3; ++i) rpid = fork();

    for (i = 0; i < 3; ++i) {
        if (rpid == 0) { // child process
            printf("PID : %d --> [%d] \n", getppid(), getpid());
            exit(10 + i);
        } else {
            printf("PID : [%d] --> %d \n", getpid(), rpid);
        }
    }

    for (i = 0; i < 3; ++i) {
        wait(&code);
        exitno = WEXITSTATUS(code);
        printf("Paraent PID = %d code = %d exitno = %d \n",
            getpid(), code, exitno);
    }
}
```

```
PID : [2331] --> 2334
PID : [2331] --> 2334
PID : [2331] --> 2334
PID : [2333] --> 2336
PID : [2333] --> 2336
PID : [2333] --> 2336
PID : [2332] --> 2337
PID : [2332] --> 2337
PID : [2332] --> 2337
PID : [2335] --> 2338
PID : [2335] --> 2338
PID : [2335] --> 2338
PID : 2332 --> [2337]
PID : 2331 --> [2334]
PID : 2333 --> [2336]
Paraent PID = 2332 code = 2560 exitno = 10
Paraent PID = 2331 code = 2560 exitno = 10
PID : 2335 --> [2338]
Paraent PID = 2333 code = 2560 exitno = 10
Paraent PID = 2333 code = 2560 exitno = 10
Paraent PID = 2333 code = 2560 exitno = 10
Paraent PID = 2335 code = 2560 exitno = 10
Paraent PID = 2335 code = 2560 exitno = 10
Paraent PID = 2335 code = 2560 exitno = 10
Paraent PID = 2331 code = 11264 exitno = 44
Paraent PID = 2332 code = 11264 exitno = 44
Paraent PID = 2332 code = 11264 exitno = 44
Paraent PID = 2331 code = 11520 exitno = 45
```

# Bad Fork Example (2)

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

int main(int argc, char* argv[])
{
    int i, rpid, code, exitno;

    for (i = 0; i < 3; ++i) rpid = fork();

    for (i = 0; i < 3; ++i) {
        if (rpid == 0) { // child process
            printf("PID : %d --> [%d] \n", getppid(), getpid());
            exit(10 + i);
        } else {
            printf("PID : [%d] --> %d \n", getpid(), rpid);
        }
    }

    for (i = 0; i < 3; ++i) {
        wait(&code);
        exitno = WEXITSTATUS(code);
        printf("Parent PID = %d code = %d exitno = %d \n",
            getpid(), code, exitno);
    }
}
```

PID : [2331] --> 2334

PID : 2331 --> [2334]

PID : [2332] --> 2337

PID : 2332 --> [2337]

PID : [2333] --> 2336

PID : 2333 --> [2336]

PID : [2335] --> 2338

PID : 2335 --> [2338]

Paraent PID = 2331 code = 2560 exitno = 10

Paraent PID = 2332 code = 2560 exitno = 10

Paraent PID = 2333 code = 2560 exitno = 10

Paraent PID = 2335 code = 2560 exitno = 10

Paraent PID = 2331 code = 11264 exitno = 44

Paraent PID = 2332 code = 11264 exitno = 44

Paraent PID = 2331 code = 11520 exitno = 45

# Bad Fork Example (3)

```
int main(int argc, char* argv[])
{
    int i, code, exitno;
    int rpid[3] = {-1, -1, -1};

    printf("PID : [%d] \n", getpid());

    if ((rpid[0] = fork()) == 0) { // child
    } else if ((rpid[1] = fork()) == 0) { // child
    } else if ((rpid[2] = fork()) == 0) { // child
    }

    for (i = 0; i < 3; ++i) {
        if (rpid[i] == 0) { // child process
            printf("PID : %d --> [%d] \n", getppid(), getpid());
            exit(10 + i);
        } else {
            printf("PID : [%d] --> %d \n", getpid(), rpid[i]);
        }
    }

    for (i = 0; i < 3; ++i) {
        wait(&code);
        exitno = WEXITSTATUS(code);
        printf("Paraent PID = %d code = %d exitno = %d \n",
            getpid(), code, exitno);
    }
}
```

```
PID : [2821]
PID : 2821 --> [2822]
PID : [2821] --> 2822
PID : [2821] --> 2823
PID : [2821] --> 2824
Paraent PID = 2821 code = 2560 exitno = 10
PID : [2823] --> 2822
PID : [2824] --> 2822
PID : [2824] --> 2823
PID : 2821 --> [2823]
PID : 2821 --> [2824]
Paraent PID = 2821 code = 2816 exitno = 11
Paraent PID = 2821 code = 3072 exitno = 12
```

# Bad Fork Example (4)

```
if ((rpid[0] = fork()) == 0) { // child
    printf("-----rpid{%d %d %d} ", rpid[0], rpid[1], rpid[2]);
    printf("PID : %d --> [%d] \n", getppid(), getpid());
    exit(10 + 0);
} else if ((rpid[1] = fork()) == 0) { // child
    printf("-----rpid{%d %d %d} ", rpid[0], rpid[1], rpid[2]);
    printf("PID : %d --> [%d] \n", getppid(), getpid());
    exit(10 + 1);
} else if ((rpid[2] = fork()) == 0) { // child
    printf("-----rpid{%d %d %d} ", rpid[0], rpid[1], rpid[2]);
    printf("PID : %d --> [%d] \n", getppid(), getpid());
    exit(10 + 2);
} else { // parents
    printf("PID : [%d] --> %d \n", getpid(), rpid[0]);
    printf("PID : [%d] --> %d \n", getpid(), rpid[1]);
    printf("PID : [%d] --> %d \n", getpid(), rpid[2]);
}

for (i = 0; i < 3; ++i) {
    wait(&code);
    exitno = WEXITSTATUS(code);
    printf("Paraent PID = %d code = %d exitno = %d \n",
        getpid(), code, exitno);
}
```

```
PID : [2459]
-----rpid{0 -1 -1} PID : 2459 --> [2460]
PID : [2459] --> 2460
PID : [2459] --> 2461
PID : [2459] --> 2462
Paraent PID = 2459 code = 2560 exitno = 10
-----rpid{2460 0 -1} PID : 2459 --> [2461]
-----rpid{2460 2461 0} PID : 2459 --> [2462]
Paraent PID = 2459 code = 2816 exitno = 11
Paraent PID = 2459 code = 3072 exitno = 12
```

# Bad Fork Example (5)

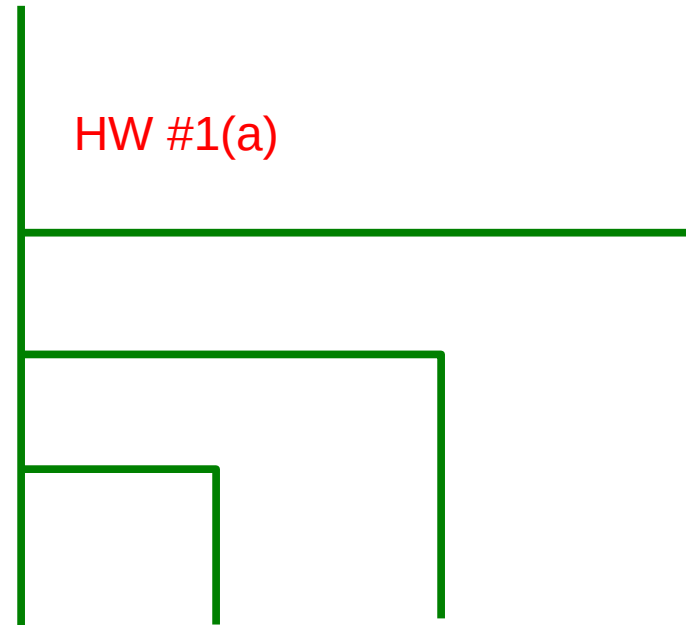
```
if ((rpid[0] = fork()) > 0) { // parent
    printf("PID : [%d] --> %d \n", getpid(), rpid[0]);

    if ((rpid[1] = fork()) > 0) { // parent
        printf("PID : [%d] --> %d \n", getpid(), rpid[1]);

        if ((rpid[1] = fork()) > 0) { // parent
            printf("PID : [%d] --> %d \n", getpid(), rpid[2]);
        } else {
            printf("-----rpid{%d %d %d} ", rpid[0], rpid[1], rpid[2]);
            printf("PID : %d --> [%d] \n", getpid(), getpid());
            exit(10 + 2);
        }

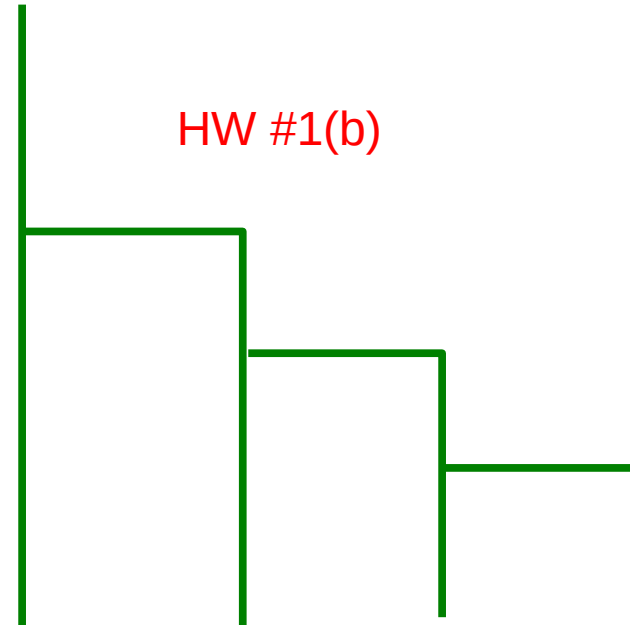
    } else {
        printf("-----rpid{%d %d %d} ", rpid[0], rpid[1], rpid[2]);
        printf("PID : %d --> [%d] \n", getpid(), getpid());
        exit(10 + 1);
    }
}

?????
```



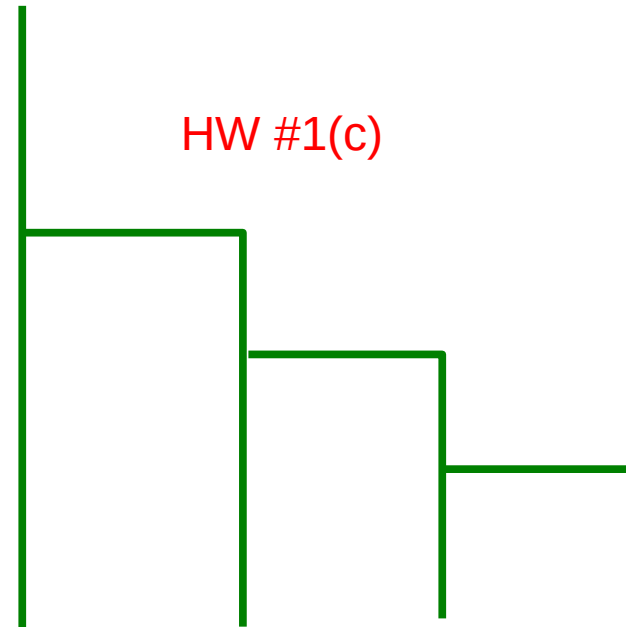
# Bad Fork Example (6)

```
printf("PID : [%d] \n", getpid());  
if ((rpid[0] = fork()) == 0) { // child  
    if ((rpid[1] = fork()) == 0) { // child  
        if ((rpid[2] = fork()) == 0) { // child  
            } else {  
            }  
        } else {  
        }  
    } else {  
    }  
}
```



# Bad Fork Example (7)

```
printf("PID : [%d] \n", getpid());  
if ((rpid[0] = fork()) > 0) { // child  
    i  
}  
} else {  
}
```



# Function Pointer

```
void f1(int x)
{
    printf("%d \n", x);
}

void f2(int x)
{
    printf("x = %d  x^2=%d \n", x, x*x);
}

int main(int argc, char* argv[])
{
    void (*foo) (int);
    int i ;

    if (argc != 2) { printf("usage: ./a.out 1 \n"); exit(0);}

    i = atoi(argv[1]);
    printf("argv[1]= %d \n", i);

    if (i==1) {    foo = f1;  }
    else if (i==2) {    foo = f2;  }

    foo(2);

}
```

# atexit()

```
void f1(void)
{
    printf("function f1() is called \n");
}

void f2(void)
{
    printf("function f2() is called \n");
}

int main(int argc, char* argv[])
{
    void (*foo) (int);
    int i ;

    atexit(f1);
    atexit(f2);
    atexit(f2);

    printf("now main function is going to be terminated \n");

    return 0;

}
```

# extern global variables

```
int l = 10;

int main(int argc, char* argv[])
{
    char *a[10];

    printf("l=%d \n", l);

    l++;
    printf("l=%d \n", l);

    f1();
    printf("l=%d \n", l);
}
```

```
extern int l ;

void f1(void) {
    printf("in f1: l=%d \n", l);
    l++;
    printf("in f1: l=%d \n", l);
}
```

# Environment

```
extern char ** environ;

int main(int argc, char* argv[])
{
    char *a[10];

    char **env = environ;

    a[0] = "abcd";

    printf("a[0] points to the char = %c \n", *(a[0]));
    printf("a[0] points to the string = %s \n", a[0]);

    printf("*env points to the char = %c \n", *(*env));
    printf("*env points to the string = %s \n", *env);

    env++;

    printf("*env points to the char = %c \n", *(*env));
    printf("*env points to the string = %s \n", *env);

}
```

# argv example

```
int main(int argc, char* argv[])
{
    int i=0;

    char **p = argv;

    printf("argv[%d]: %s \n", i, *p);
    i++; p++;

    printf("argv[%d]: %s \n", i, *p);
    i++; p++;

    printf("argv[%d]: %s \n", i, *p);
    i++; p++;

}
```

# Reference

---

## References

- [1] <http://en.wikipedia.org/>
- [2] S.S. Park, Linux Practical Command Bible (in Korean)
- [3] S.S. Park, Linux Server Practical Administration Bible (in Korean)