Process (1A)

- Process Command

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Young Won Lim
11/15/2012

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 #

background

child process

pstree

```
gnome-terminal─┬─bash─┬─pstree
               │      └─xterm───bash───octave
               ├─gnome-pty-helpe
               └─3*[{gnome-terminal}]
```

chmod chown chgrp

ps -ef

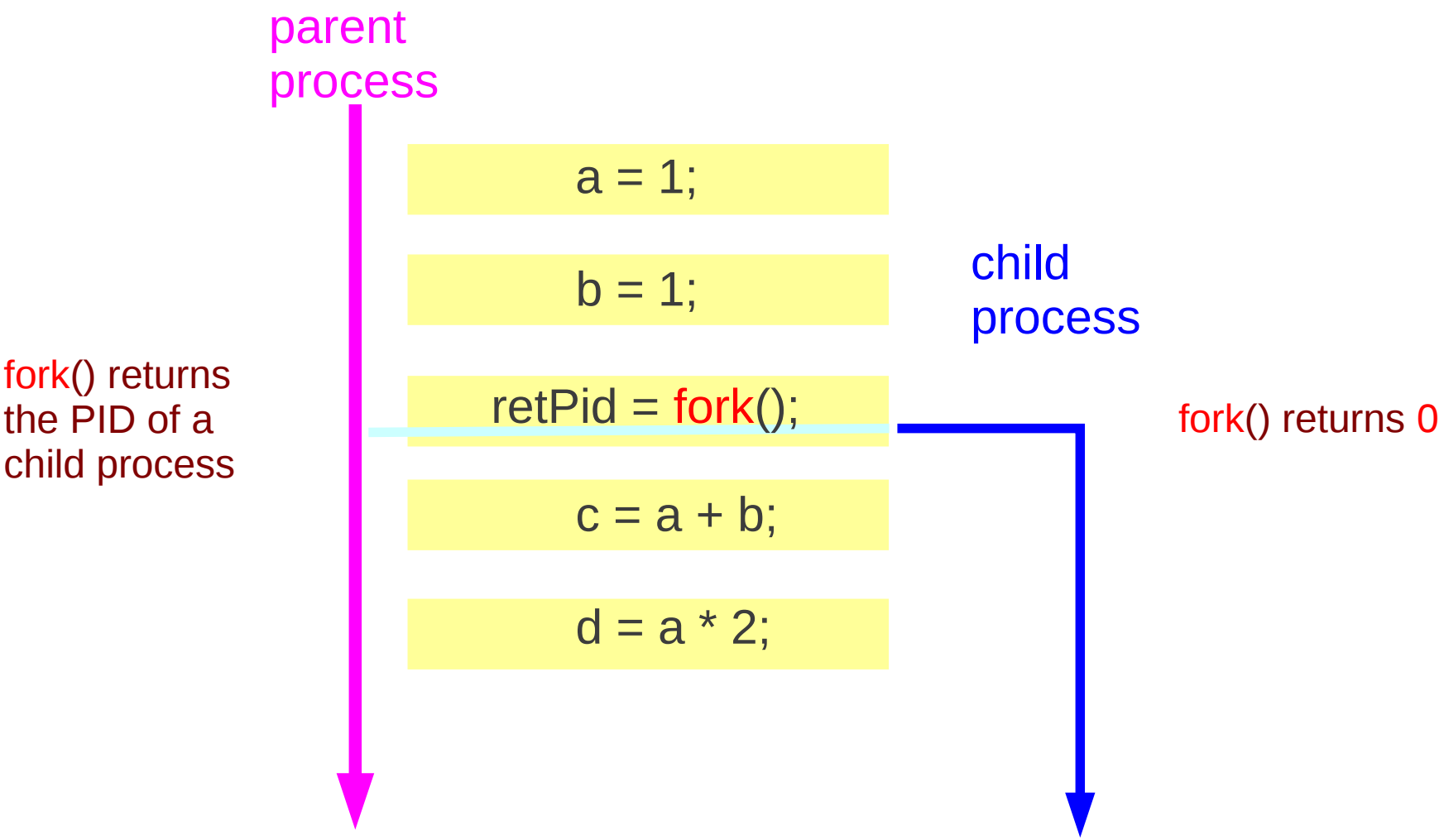
ps -ef

```
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()


parent
process
a = 1;
b = 1;
retPid = fork();
c = a + b;
d = a * 2;
child
process
fork() returns
the PID of a
child process
fork() returns 0

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";

      execl("/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 bye (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
```

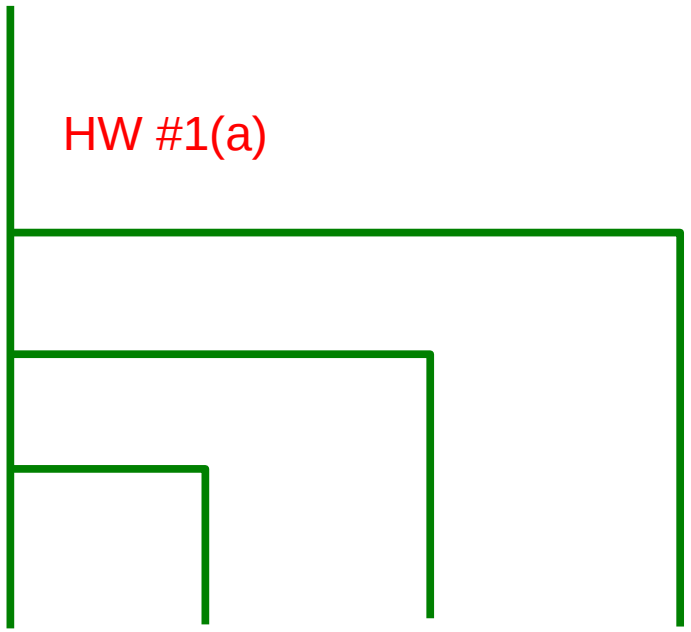
Bad Fork Example (4)

```
  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
```

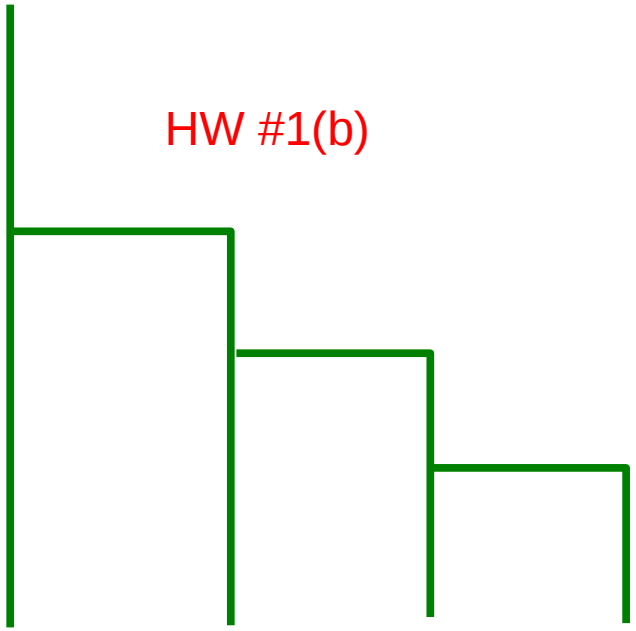
Bad Fork Example (5)

```
   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", getppid(), getpid());
           exit(10 +2);
         }

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



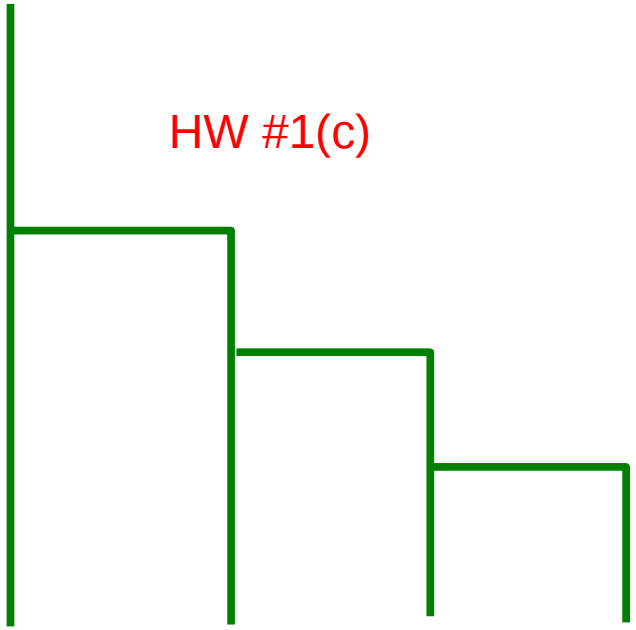
Bad Fork Example (6)

```
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("agrv[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 I = 10;

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


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

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

   f1();
   printf("I=%d \n", I);

}
```

```
extern int I ;

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

}
```

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)