Tutorial 9
NWEN241
Process Management System Calls

Kirita-Rose Escott

kirita-rose.escott@ecs.vuw.ac.nz

Content

- Process management system calls
  - `fork()`
  - `exec()`

    Defined in unistd.h

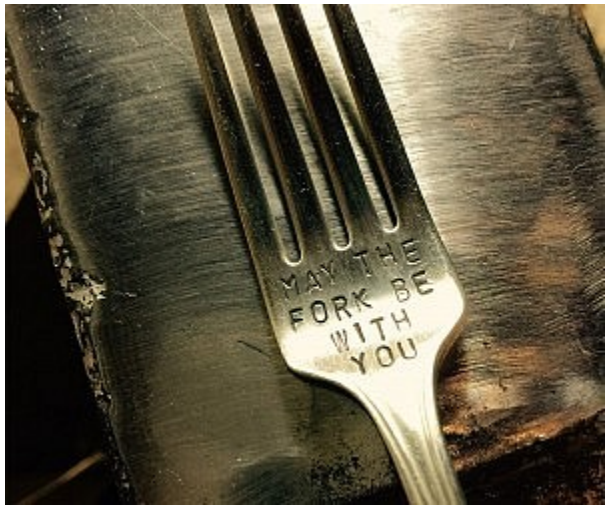
  - `wait()`

    Defined in sys/wait.h

  - `exit()`

    Defined in stdlib.h

Process creation with **`fork()`**



- A process calling **`fork()`** spawns a child process.

- After a successful **`fork()`** call, two copies of the original code will be running.
  - Parent process – *return value* of `fork()` → *child PID*.
  - New child process – *return value* of `fork()` → 0.

Example: fork1.c

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

int main(void) {
pid_t pid;

printf("fork test\n");
  if ((pid = fork ()) < 0) {
     printf(" fork error\n");    }
  else if (pid == 0) { /* child */
    printf(" at child process\n");    }
 else { /* parent */
    printf(" at parent process\n");
    wait(NULL);    }
 exit(0);
}
```

Example: fork2.c

```
printf("fork test\n");
  if ((pid = fork ()) < 0) {
      printf(" fork error\n");
} else if (pid == 0) { /* child */
     printf(" at child process\n");
} else { /* parent */
     printf(" at parent process\n");
     wait(NULL);
}
 printf(" where am i?\n");
 exit(0);
```

How do we know where we are?

Example: fork3.c

```
printf("fork test\n");
  if ((pid = fork ()) < 0) {
      printf(" fork error\n");    }
  else if (pid == 0) { /* child */
     printf(" at child process, my pid is: %d\n", (int) getpid());}
 else { /* parent */
     printf(" at parent process, my pid is: %d\n", (int) getpid());
     wait(NULL);    }
 printf(" i am here with pid : %d\n", (int) getpid());
 exit(0);
```

Example: fork4.c

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

int main(void){
  FILE *fp = fopen("xxx.txt", "a");
  pid_t p = fork();

  if(p < 0) { /* Failed */
    exit(0);
  } else if(p == 0) { /* Child */
    fputs("child\n", fp);
  } else {  /* Parent */
    fputs("parent\n", fp);
  }
}
```

Example: fork5.c

```
int main(void) {
  pid_t pid;
  int a = 2; int b = 4;

  printf("fork test\n");
  printf("a = %d and b = %d\n \n", a, b

  if ((pid = fork ()) < 0) {
    printf(" fork error\n");
  } else if (pid == 0) { /* child */
    a = a * 2;          b = b * 2;
    printf(" at child process, a = %d and b = %d\n", a, b);
  } else { /* parent */
     a = a + 5;          b = b + 5;
    printf(" at parent process, a = %d and b = %d\n", a, b);
    wait(NULL);
}
exit(0);
}
```

What happens when both process have access to a and b?

`wait()` Call System

- Forces the parent to suspend execution, i.e. wait for its children or a specific child to die (*terminate* is more appropriate terminology, but a bit less common).

```
pid_t wait(int *status);
```

- The status, if not NULL, stores exit information of the child, which can be analyzed by the parent.
- The return value is:
  - PID of the exited process, if no error
  - (-1) if an error has happened

`exit()` System Call

- Gracefully terminates process execution, meaning it does clean up and release of resources, and puts the process into the ***zombie*** state → *terminated but still waiting for parent process to read its exit status*.
  - By calling **`wait()`**, the parent cleans up all its zombie children.

- **`exit()`** specifies a return value from the program, which a parent process might want to examine as well as status of the dead process.

Example: wait1.c

```
int main(void){
  FILE *fp = fopen("xxx.txt", "a");
  pid_t p = fork();

  if(p < 0) { /* Failed */
    exit(0);
  } else if(p == 0) { /* Child */
    fputs("child says: hi\n", fp);
    fputs("child says: I am\n", fp);
    fputs("child says: the child\n", fp);
  } else {  /* Parent */
    wait(NULL);
    fputs("parent says: hello\n", fp);
}
}
```

What happens when we don’t exit the child process? – Example: fork6.c

```
int main(void) {
pid_t pid;

printf("fork test\n");
  if ((pid = fork ()) < 0) {
     printf(" fork error\n");
} else if (pid == 0) { /* child */
    printf(" at child process\n");
    while(1);
} else { /* parent */
    printf("parent: wait for child\n");
    wait(NULL);
    printf("parent: child complete\n");
}
exit(0);
}
```

More about `wait()` and `exit()`

- Should not interpret the status value of system call **`wait(&status)`** literally. If **`&status`** is not NULL, **`wait()`** stores status information in the **`int`** to which it points.
- Value returned by **`exit(&status)`** is moved to 2nd byte and 1st (lowest) byte is used to store the status information.
- In previous example:

```
    scanf(" %d", &rv); // if value of x is entered
    ...
    exit(rv);
...
    wait(&rv); // the rv contents will be x left shift
               // by 8 bits and additional status
               // written into lowest 8 bit
```

0	0	0	x

0	0	x	status

Example: waitexit.c

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

main()
{
  int pid; int rv;

  pid=fork();
  switch(pid){
    case -1:
      printf("Error -- Something went wrong with fork()\n");
      exit(1); // parent exits
    case 0:
      printf("CHILD: This is the child process!\n");
      printf("CHILD: My PID is %d\n", getpid());
      printf("CHILD: My parent's PID is %d\n", getppid());
      printf("CHILD: Enter my exit status: ");
      scanf(" %d", &rv);
      printf("CHILD: I'm outta here!\n");
      exit(rv);
    default:
      printf("PARENT: This is the parent process!\n");
      printf("PARENT: My PID is %d\n", getpid());
      printf("PARENT: My child's PID is %d\n", pid);
      printf("PARENT: I'm now waiting for my child to exit()...\n");
      wait(&rv);
      printf("PARENT: My child's exit status is 0x%.8X\n", rv);
      printf("PARENT: I'm outta here!\n");
  }
}
```

Process creation with **exec()**

- There is no system call specifically by the name **exec()**
- By **exec()** we usually refer to a family of calls:
  - `int execl(char *path, char *arg, ...);`
  - `int execv(char *path, char *argv[]);`
  - `int execle(char *path, char *arg, ..., char *envp[]);`
  - `int execve(char *path, char *argv[], char *envp[]);`
  - `int execlp(char *file, char *arg, ...);`
  - `int execvp(char *file, char *argv[]);`
- The various options *l*, *v*, *e*, and *p* mean:
  - ***l*** : an argument list,
  - ***v*** : an argument vector,
  - ***e*** : an environment vector, and
  - ***p*** : a search path.

Example: execl.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  int ret;

  printf("Calling execl...\n");
  ret = execl("ls", "ls", NULL);
  printf("Failed execl... ret = %d\n", ret);

 return 0;
}
```

Why does this fail?

Example: corrected execl.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  int ret;

  printf("Calling execl...\n");
  ret = execl("/bin/ls", "ls", NULL);
  printf("Failed execl... ret = %d\n", ret);

 return 0;
}
```

use 'which ls' in the cmd to find the path

Example: execv.c

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

int main(void) {
  int ret;
  char *args[] = {"hello", "one", "two", "three", NULL};

  printf("Calling execv...\n");
  ret = execv("./output1", args);
  printf("Failed execv... ret = %d\n", ret);

  exit(0);
}
```

What is ./output1?

Example: output1.c

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

int main(int argc, char *argv[]) {
   int i;
   printf("ARGUMENTS:\n");

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

Example: execve.c

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

int main(void) {
  int ret;
  char *args[] = {"hello", "one", "two", "three", NULL};
 char *envp[] = {"some", "environment", "variables", "ok", NULL};

  printf("Calling execve..\n");
  ret = execv("./output2", args, envp);
  printf("Failed execve... ret = %d\n", ret);

  exit(0);
}
```

Example: output2.c

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

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

   printf("ARGUMENTS:\n");

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

  printf("ENVIRONMENT VARIABLES:\n");

  for  (j=0;  j<argc;  j++)        {
      printf("%s\n",  envp[j]);
}
   return 0;
}
```

Example: execlp.c

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

int main(void) {
pid_t pid;

printf("fork test\n");
  if ((pid = fork ()) < 0) {
    printf(" fork error\n");
  } else if (pid == 0) { /* child */
    printf("child: execute ls command\n");
    execlp("/bin/ls", "ls", NULL);
  } else { /* parent */
    printf("parent: wait for child\n");
    wait(NULL);
    printf("parent: child complete\n");
}
exit(0);
}
```