Linux Systems Programming
Process Handling

CSE 410- Project01

How to call and create a process

- exec command family
  - **execlp** (*path, arg0, arg1, ...*)
    - This is equivalent to "execvp(*path*, (*arg0*, *arg1*, ...))".
  - **execvp** (*path, args*)
    - This is like "execv(*path*, *args*)" but duplicates the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from environ['PATH'];
  - and similar functions such as **execl, execle, execve, execvp** , **execvpe**

Example:

```
char *argv[20];
…
execvp(argv[0], argv);
```

How to call and create a process (calling an executable in program)

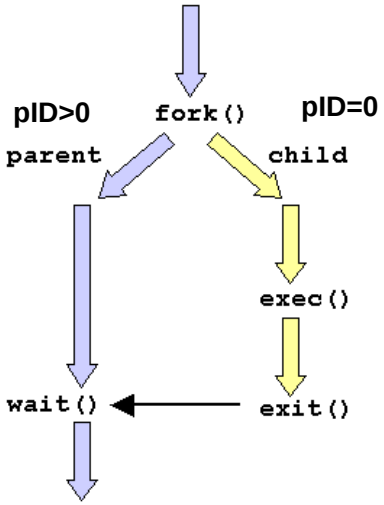
This example take the list of the files from the current directory:

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

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

   argv[0] = "ls";
   argv[1] = "-l";
   argv[2] = NULL;

   execvp("ls", argv);
   cout << "command not found" << endl;
}
```

Creating process

- New processes are created with the fork() system call
  - pid_t pID = fork();
  - The return value (pID) of the function is which discriminates the two threads of execution. A zero is returned by the fork function in the child's process.

```
if (fork()) {
  /* Do the parent tasks */
} else {
  /* Do the child tasks */
}
```



fork() example (1)

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

int main(){
   int x = 5;
   int pid = fork();
   //////the second process (child) is instantiated//////
   cout <<"This line is repeated!"<<endl;
   /////Distinguishing between parent and child tasks////
   if (pid > 0) {// in parent process
          x = 6;
          cout << "ID of child is " << pid << endl;
   }else{  //in child process
          cout << "In child x = " << x << endl;
   }
}
```

fork() example (2)

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream.h>
int main(){
   int pid = fork();
   if (pid > 0)  {
      cout << "I'm Parent" << endl;
      sleep(1);
   } else   {
   cout << "I am Child" << endl;
   char *argv[10];
   argv[0] = "ls";
   argv[1] = "-l";
   argv[2] = NULL;
   argv[3] = NULL;

   execvp("ls", argv);
   cout << “command not found" << endl;
   }
}
```

Using fork() & exec() within a loop

```
while(contLoop)
{
        switch( pid = fork() )
        {
                case -1:    // fork error
                            break;

                case 0:     // child process
                            err = exec();           // some exec call
                            // print error message here
                            exit();
                            break;                  // keep compiler happy

                default:    // parent process
                            // do parent stuff
                            break;
        }  // switch
}  // while
```

wait() and exit() function

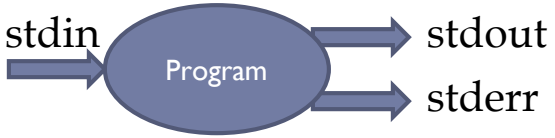
- The parent process will often want to wait until all child processes have been completed.
  - **wait(int *status)**: Blocks calling process until the child process terminates. If child process has already terminated, the wait() call returns immediately. If the calling process has multiple child processes, the function returns when one returns.
  - **exit(int status):** Causes normal process termination and the value of status is returned to the parent
  - **waitpid():** Options available to block calling process for a particular child process not the first one.
    - (for this project you actually do not this one)

I/O Redirection

- Output redirection: redirecting the printed output from *stdout* to specific file
  - Example: ls -al > myOutputFile
- Input redirection: redirecting the read input from *stdin* to a specific file
  - Example: sort -n <myInputFile

The Unix standard I/O streams are:

Handle	Name
0	stdin
1	stdout
2	stderr



Some combination examples:

cat -n < inFile > outFile

Output Redirection: example code for: ls -l >temp

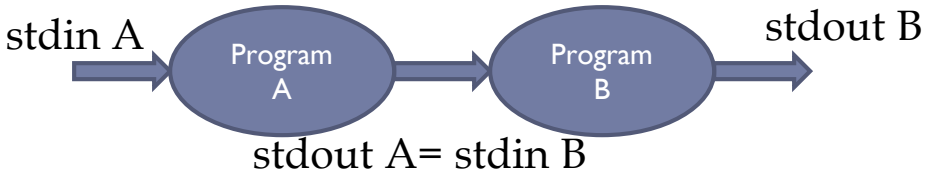
```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/wait.h>
int main(int argc, char *argv[])
{
    char *path = "/bin/ls";
    char *arg0 = "ls";
    pid_t pid;
    int fd;
    int status;
    pid = fork();
    if (pid == 0) // child
    {
        fd = open("temp", O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU);
        dup2(fd, STDOUT_FILENO);                                    ←duplicate file descriptor 'fd' to stdout
        close(fd);                                                            Note that STDOUT_FILENO=1
        if (execl(path, arg0, NULL) == -1)
            perror("execl");
         exit;
     }
     else      // parent
     {
        close(fd);
        wait(&status);
    }
}
```

Input Redirection: example code for: sort <temp

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/wait.h>
int main(int argc, char *argv[])
{
    char *path = "/usr/bin/sort";
    char *arg0 = "sort";
    pid_t pid;
    int fd;
    int status;
    pid = fork();
    if (pid == 0) // if child…
    {
        fd = open("temp", O_RDWR);
        dup2(fd, STDIN_FILENO);                    ← dup2, STDIN now points to same file descriptor as ‘fd’
        close(fd);                                                    Note that STDIN_FILENO=0
        if (execl(path, arg0, NULL) == -1)
        {
            perror("execl");
        }
        exit();
    }
    else         // parent…
    {
        close(fd);
        wait(&status);
    }
}
```

Piping

- Piping: Programs can be run together such that one program reads the output from another with no need for an explicit intermediate file:
  - command1 | command2
  - ps -ax | wc -l



Example: ls -l | grep myfile | wc -l

Piping:
example Code for : ls -l | wc

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream.h>
int  main()
{
       int fd[2];
       pipe(fd);                                              ← piping two file descriptors
       int pid = fork();
       if (pid > 0)             // child…
       {
         close(fd[1]);
         int ret = dup2(fd[0],0);                             ←duplicate std input to file descriptor fd[0]
         if (ret < 0) perror("dup2");
         char *argv[10];
         argv[0] = "wc";  argv[1] = NULL;
         execvp("wc", argv);
         exit();
      }
      else                     //…parent
      {
         int ret = dup2(fd[1],1);                             ←duplicate std output to file descriptor fd[1]
         if (ret < 0)
         {
            perror("dup2");
         }
         char *argv[10];
         argv[0] = "ls";    argv[1] = "-l“;   argv[2] = NULL;
         execvp("ls", argv);
      }
}
```

Listing Process Environmental Variables

```
int main(int argc, char **argv, char **envp)
{
        for (char **env = envp; *env != 0; env++)
        {
                char *thisEnv = *env;
                printf("%s\n", thisEnv);
        }
        return 0;
}
```

*** sample code from: *stackoverflow.com*

Note: Specific environmental variables can be viewed/modified with getenv() and setenv() respectively.