Example of using Fork and pipe in Linux

Posted by Alejandro G. Carlstein Ramos Mejia on October 16, 2010 October 16, 2010 About Programming / ANSI/POSIX C / Linux

The following program will create a chain of parents/child in which the last child will receive a message form the top parent through a pipe.

NOTIFICATION: These examples are provided for educational purposes. Using this code is under your own responsibility and risk. The code is given 'as is'. I do not take responsibilities of how they are used.

The first example is without using recursion while the second example is the same thing but using example. In this way, you can see how fork and pipe behaves.

for_pipe_without_using_recursion.c:

```
/**
 * @author: Alejandro G. Carlstein R. M.
 * @description: This program will take an integer argument.
 *               It will create a chain of parent-child equal to the value
 *               provided by the integer argument, Example:
 *     Top parent - forks ==>
 *      child1 - forks ==>
 *     child1's child - forks ... ==> Nth generation child, where
 *     N is the integer argument.
 *        Then the top parent will then pass a message to
 *               the child at the lowest level (Nth generation child)
 *               through a pipe. That child will print
 *               the received message to output stdout.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#define MAX_N 3
#define IDX_ARG_N 1
#define MAX_BUFF 30

int read_pipe(int *pfds, char msg[], int max_buff, int i_tab);

int write_pipe(int *pfds, const char *msg, int max_buff, int i_tab);

int main(int argc, char *argv[])
{
```

```
pid_t rtn_pid, pid[3];

int rtn_error = 0;

int i, j, status;

int pfds[2];

int i_N = (argc > 2) ? atoi(argv[IDX_ARG_N]): 0;

char buf[MAX_BUFF];

printf('\n');

/* create a pipe */
   if (pipe(pfds) == -1) {
           perror('pipe');
           exit(1);
   }else{

 /* fork first child */
    if ( (pid[0] = fork()) < 0) {
  perror('fork pid[0]');
  exit(1);
    }// end if

    /* Parent of first fork */
    if (pid[0] > 0){

     printf('Parent of first fork sending message: Hello! \n');

     rtn_error = write_pipe(pfds, 'Hello! \n', MAX_BUFF, 0);

     //close(pfds[0]);
     //close(pfds[1]);

    }//end if

    /* Child of first fork */
    if (pid[0] == 0){

     printf('>>Child of first fork \n');

  /* fork second child */
     if ( (pid[1] = fork()) < 0) {
   perror('fork pid[1]');
   exit(1);
     }// end if

     /* Parent of second fork */
     if (pid[1] > 0){
```

```
       //printf('Parent of second fork...\n');

   }//end if

      /* Child of second fork */
      if (pid[1] == 0){

       printf('>>>>Child of second fork \n');

    /* fork third child */
       if ( (pid[2] = fork()) < 0) {
     perror('fork pid[2]');
     exit(1);
       }// end if

       /* Parent of third fork */
       if (pid[2] > 0){

        //printf('Parent of third fork...\n');

    }//end if

       /* Child of third fork */
       if (pid[2] == 0){

        printf('>>>>>Child of third fork \n');

     rtn_error = read_pipe(pfds, buf, MAX_BUFF, 5);

     if (rtn_error == 0){
      printf('>>>>>CHILD OF THIRD FORK READ MESSAGE: %s \n',  buf);
     }//end if

       }//end if

      }//end if

     }//end if

  wait();

 }// end if

 return rtn_error;
}

int write_pipe(int *pfds, const char *msg, int max_buff, int i_tab){

  int rtn_error = 0;

  /* close the read end */
  close(pfds[0]);
```

```
  //printf ('%*sWRITE_PIPE: %s\n', i_tab, ' ', msg);

  /* write message to parent  through the write end */
        if(write(pfds[1], msg, max_buff) <= 0) {

   printf('%*s[X] ERROR: Cannot write\n', i_tab, ' ');

   rtn_error = 1;

  }//end if

 return rtn_error;
}

int read_pipe(int *pfds, char msg[], int max_buff, int i_tab){

  int rtn_error = 0;

  /* close the write end */
  close(pfds[1]);

  //printf ('%*sREAD_PIPE: ', i_tab, ' ');

  /* read message from child through the read end */
  // If not data in the pipe, the read will block
     if( read(pfds[0], msg, max_buff) <= 0 ) {

   printf ( '%*s[X] ERROR: Cannot read\n', i_tab, ' ');

   rtn_error = 1;

  }else{

   //printf('%*s%s \n', i_tab, ' ', msg);

  }//end if

 return rtn_error;
}
```

fork_pipe_using_recursion.c:

```
/**
 * @author: Alejandro G. Carlstein R. M.
 * @description: This program will take an integer argument.
 *               It will create a chain of parent-child equal to the value
 *               provided by the integer argument, Example:
 *     Top parent - forks ==>
 *      child1 - forks ==>
 *     child1's child - forks ... ==> Nth generation child, where
 *     N is the integer argument.
 *        Then the top parent will then pass a message to
 *               the child at the lowest level (Nth generation child)
```

```
 *               through a pipe. That child will print
 *               the received message to output stdout.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#define MAX_N 3
#define IDX_ARG_N 1
#define MAX_BUFF 30
#define DBG_LV1 0
#define DBG_LV2 1
#define MSG 'Hello! \n'

int read_pipe(int *pfds, char msg[], int max_buff, int i_tab);

int write_pipe(int *pfds, const char *msg, int max_buff, int i_tab);

int recursive_f(int *pfds, int i_count, int i_max_count);

int exec_order;

int main(int argc, char *argv[])
{
 pid_t rtn_pid, pid[3];

 int rtn_error = 0;

 int i, status;

 int pfds[2];

 int i_N = (argc > 2) ? atoi(argv[IDX_ARG_N]): 0;

 exec_order = 0;

 printf('\n');

 /* create a pipe */
    if (pipe(pfds) == -1) {
            perror('pipe');
            exit(1);
    }else{

     rtn_error = recursive_f(pfds, MAX_N, MAX_N);

 }// end if

 close(pfds[0]);
```

```
 close(pfds[1]);

 for (i = 0; i < MAX_N - 1; i++){
  wait();
 }

 return rtn_error;
}

int recursive_f(int *pfds, int i_count, int i_max_count){

 int rtn_error = 0;

 char buf[MAX_BUFF];

 pid_t pid;

 if (DBG_LV1){

     printf('[e_o: %d][count: %d/%d] recursive_f (*pfds, %d, %d)\n',
        exec_order, i_count, i_max_count, i_count, i_max_count);

 }//end if

 /* fork */
    if ( ( pid = fork() ) < 0) {

  if (DBG_LV2)
   perror('error fork pid');

  rtn_error = 1;

    }else{

     /* Parent of first fork */
     if (pid > 0 && i_count == i_max_count){

      if (DBG_LV1)
       printf('[e_o: %d][count: %d/%d] ', exec_order, i_count, i_max_count);

      printf('Parent of first fork sending message: %s', MSG);

      rtn_error = write_pipe(pfds, MSG, MAX_BUFF, 0);

      if (DBG_LV1)
       printf('[e_o: %d][count: %d/%d] ', exec_order, i_count, i_max_count);

      printf('PARENT OF FIRST FORK SENT MESSAGE: %s', MSG);

     }else{
      exec_order++;

     }//end if
```

```
  /* Child of last fork */
  if (pid == 0){

   if (i_count < 1){

       if (DBG_LV1)
        printf('[e_o: %d][count: %d/%d] ', exec_order, i_count, i_max_count);

    printf('Last Child of fork reading...\n');

    rtn_error = read_pipe(pfds, buf, MAX_BUFF, 5);

    if (rtn_error == 0){

        if (DBG_LV1)
         printf('[e_o: %d][count: %d/%d] ', exec_order, i_count, i_max_count);

     printf('LAST CHILD OF FORK READ MESSAGE: %s', buf);

    }//end if

   }else{

      i_count--;

      if (DBG_LV1)
       printf('[e_o: %d][count: %d/%d] Calling recursive_f (i_count[%d] and
i_max_count[%d]) \n',
           exec_order, i_count, i_max_count, i_count, i_max_count);

      rtn_error = recursive_f(pfds, i_count, i_max_count);

     }//end if

     }//end if

    }// end if

 return rtn_error;
}

int write_pipe(int *pfds, const char *msg, int max_buff, int i_tab){

  int rtn_error = 0;

  /* close the read end */
  close(pfds[0]);

  if (DBG_LV1) printf ('%*sWRITE_PIPE: %s\n', i_tab, ' ', msg);

  /* write message to parent  through the write end */
       if(write(pfds[1], msg, max_buff) <= 0) {
```

```
   if (DBG_LV2) printf('%*s[X] ERROR: Cannot write\n', i_tab, ' ');

   rtn_error = 1;

  }//end if

 return rtn_error;
}

int read_pipe(int *pfds, char msg[], int max_buff, int i_tab){

  int rtn_error = 0;

  /* close the write end */
  close(pfds[1]);

  if (DBG_LV1) printf ('%*sREAD_PIPE: ', i_tab, ' ');

  /* read message from child through the read end */
  // If not data in the pipe, the read will block
     if( read(pfds[0], msg, max_buff) <= 0 ) {

   if (DBG_LV2) printf ( '%*s[X] ERROR: Cannot read\n', i_tab, ' ');

   rtn_error = 1;

  }else{

   if (DBG_LV1) printf('%*s%s \n', i_tab, ' ', msg);

  }//end if

 return rtn_error;
}
```

If you encounter any problems or errors, please let me know by providing an example of the code, input, output, and an explanation. Thanks..

```
/**
* @course: CS350 Lab
* @author: Alejandro G. Carlstein R. M.
* @description: This program will take an integer argument.
*          It will create a chain of parent-child equal to the value
*          provided by the integer argument, Example:
*            Top parent – forks ==>
*             child1 – forks ==>
*            child1's child – forks ... ==> Nth generation child, where
*            N is the integer argument.
*            Then the top parent will then pass a message to
```

```
*           the child at the lowest level (Nth generation child)
*           through a pipe. That child will print
*           the received message to output stdout.
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#define MAX_N 3
#define IDX_ARG_N 1
#define MAX_BUFF 30
#define DBG_LV1 0
#define DBG_LV2 1
#define MSG “Hello! \n”

int read_pipe(int *pfds, char msg[], int max_buff, int i_tab);

int write_pipe(int *pfds, const char *msg, int max_buff, int i_tab);

int recursive_f(int *pfds, int i_count, int i_max_count);

int exec_order;

int main(int argc, char *argv[])
{
pid_t rtn_pid, pid[3];

int rtn_error = 0;

int i, status;

int pfds[2];

int i_N = (argc > 2) ? atoi(argv[IDX_ARG_N]): 0;

exec_order = 0;

printf(“\n”);

/* create a pipe */
if (pipe(pfds) == -1) {
perror(“pipe”);
exit(1);
}else{
```

```
rtn_error = recursive_f(pfds, MAX_N, MAX_N);

}// end if

close(pfds[0]);
close(pfds[1]);

for (i = 0; i < MAX_N – 1; i++){
wait();
}

return rtn_error;
}

int recursive_f(int *pfds, int i_count, int i_max_count){

int rtn_error = 0;

char buf[MAX_BUFF];

pid_t pid;

if (DBG_LV1){

printf("[e_o: %d][count: %d/%d] recursive_f (*pfds, %d, %d)\n",
exec_order, i_count, i_max_count, i_count, i_max_count);

}//end if

/* fork */
if ( ( pid = fork() ) < 0) {

if (DBG_LV2)
perror("error fork pid");

rtn_error = 1;

}else{

/* Parent of first fork */
if (pid > 0 && i_count == i_max_count){

if (DBG_LV1)
printf("[e_o: %d][count: %d/%d] ", exec_order, i_count, i_max_count);

printf("Parent of first fork sending message: %s", MSG);

rtn_error = write_pipe(pfds, MSG, MAX_BUFF, 0);
```

```
if (DBG_LV1)
printf("[e_o: %d][count: %d/%d] ", exec_order, i_count, i_max_count);

printf("PARENT OF FIRST FORK SENT MESSAGE: %s", MSG);

}else{
exec_order++;

}//end if

/* Child of last fork */
if (pid == 0){

if (i_count < 1){

if (DBG_LV1)
printf("[e_o: %d][count: %d/%d] ", exec_order, i_count, i_max_count);

printf("Last Child of fork reading...\n");

rtn_error = read_pipe(pfds, buf, MAX_BUFF, 5);

if (rtn_error == 0){

if (DBG_LV1)
printf("[e_o: %d][count: %d/%d] ", exec_order, i_count, i_max_count);

printf("LAST CHILD OF FORK READ MESSAGE: %s", buf);

}//end if

}else{

i_count–;

if (DBG_LV1)
printf("[e_o: %d][count: %d/%d] Calling recursive_f (i_count[%d] and i_max_count[%d]) \n",
exec_order, i_count, i_max_count, i_count, i_max_count);

rtn_error = recursive_f(pfds, i_count, i_max_count);

}//end if

}//end if

}// end if

return rtn_error;
}

int write_pipe(int *pfds, const char *msg, int max_buff, int i_tab){
```

```
int rtn_error = 0;

/* close the read end */
close(pfds[0]);

if (DBG_LV1) printf ("%*sWRITE_PIPE: %s\n", i_tab, " ", msg);

/* write message to parent  through the write end */
if(write(pfds[1], msg, max_buff) <= 0) {

if (DBG_LV2) printf("%*s[X] ERROR: Cannot write\n", i_tab, " ");

rtn_error = 1;

}//end if

return rtn_error;
}

int read_pipe(int *pfds, char msg[], int max_buff, int i_tab){

int rtn_error = 0;

/* close the write end */
close(pfds[1]);

if (DBG_LV1) printf ("%*sREAD_PIPE: ", i_tab, " ");

/* read message from child through the read end */
// If not data in the pipe, the read will block
if( read(pfds[0], msg, max_buff) <= 0 ) {

if (DBG_LV2) printf ( "%*s[X] ERROR: Cannot read\n", i_tab, " ");

rtn_error = 1;

}else{

if (DBG_LV1) printf("%*s%s \n", i_tab, " ", msg);

}//end if

return rtn_error;
}
```