Subsections

Interprocess Communication (IPC), Pipes

We have now began to see how multiple processes may be running on a machine and maybe be controlled (spawned by `fork()` by one of our programs.

In numerous applications there is clearly a need for these processes to communicate with each exchanging data or control information. There are a few methods which can accomplish this task. We will consider:

- Pipes
- Signals
- Message Queues
- Semaphores
- Shared Memory
- Sockets

In this chapter, we will study the piping of two processes. We will study the others in turn in subsequent chapters.

Piping in a C program: <stdio.h>

Piping is a process where the input of one process is made the input of another. We have seen examples of this from the UNIX command line using |.

We will now see how we do this from C programs.

We will have two (or more) `forked` processes and will communicate between them.

We must first open a ***pipe***

UNIX allows two ways of opening a pipe.

popen() -- Formatted Piping

`FILE *popen(char *command, char *type)` -- opens a pipe for I/O where the command is the process that will be connected to the calling process thus creating the ***pipe***. The type is either ``r'' - for reading, or ``w'' for writing.

`popen()` returns is a stream pointer or NULL for any errors.

A pipe opened by `popen()` should always be closed by `pclose(FILE *stream)`.

We use `fprintf()` and `fscanf()` to communicate with the pipe's `stream`.

`pipe()` -- Low level Piping

`int pipe(int fd[2])` -- creates a pipe and returns two file descriptors, `fd[0], fd[1]`. `fd[0]` is opened for reading, `fd[1]` for writing.

`pipe()` returns 0 on success, -1 on failure and sets `errno` accordingly.

The standard programming model is that after the pipe has been set up, two (or more) cooperative processes will be created by a fork and data will be passed using `read()` and `write()`.

Pipes opened with `pipe()` should be closed with `close(int fd)`.

Example: Parent writes to a child

```
int pdes[2];

pipe(pdes);
if ( fork() == 0 )
  { /* child */
                                  close(pdes[1]);
                                  read( pdes[0]); /* read from parent */
                                  .....
                  }
else
                  {               close(pdes[0]);
                                  write( pdes[1]); /* write to child */
                                  .....
                  }
```

An futher example of piping in a C program is `plot.c and subroutines and it performs as follows:`

- `The program has two modules plot.c (main) and plotter.c.`
- `The program relies on you having installed the freely` ***`gnuplot`*** `graph drawing program in the directory /usr/local/bin/ (in the listing below at least) -- this path could easily be changed.`
- `The program plot.c calls` ***`gnuplot`***
- `Two Data Stream is generated from Plot`
  - $y = sin(x)$
  - $y = sin(1/x)$
- `2 Pipes created -- 1 per Data Stream.`
- `°` ***`Gnuplot`*** `produces ``live'' drawing of output.`

`The code listing for plot.c is:`

```
/* plot.c - example of unix pipe. Calls gnuplot graph drawing package to draw
   graphs from within a C program. Info is piped to gnuplot */
/* Creates 2 pipes one will draw graphs of y=0.5 and y = random 0-1.0 */
/* the other graphs of y = sin (1/x) and y = sin x */

/* Also user a plotter.c module */
/* compile: cc -o plot plot.c plotter.c */

#include "externals.h"
#include <signal.h>

#define DEG_TO_RAD(x) (x*180/M_PI)

double drand48();
void quit();
```

```
FILE *fp1, *fp2, *fp3, *fp4, *fopen();

main()
{   float i;
    float y1,y2,y3,y4;

    /* open files which will store plot data */
    if ( ((fp1 = fopen("plot11.dat","w")) == NULL) ||
           ((fp2 = fopen("plot12.dat","w")) == NULL) ||
            ((fp3 = fopen("plot21.dat","w")) == NULL) ||
             ((fp4 = fopen("plot22.dat","w")) == NULL) )
              { printf("Error can't open one or more data files\n");
                exit(1);
              }

    signal(SIGINT,quit); /* trap ctrl-c call quit fn */
    StartPlot();
    y1 = 0.5;
    srand48(1); /* set seed */
    for (i=0;;i+=0.01) /* increment i forever use ctrl-c to quit prog */
      { y2 =  (float) drand48();
        if (i == 0.0)
           y3 = 0.0;
       else
           y3 = sin(DEG_TO_RAD(1.0/i));
        y4 = sin(DEG_TO_RAD(i));

        /* load files */
        fprintf(fp1,"%f %f\n",i,y1);
        fprintf(fp2,"%f %f\n",i,y2);
        fprintf(fp3,"%f %f\n",i,y3);
        fprintf(fp4,"%f %f\n",i,y4);

        /* make sure buffers flushed so that gnuplot */
        /*  reads up to data file */
        fflush(fp1);
        fflush(fp2);
        fflush(fp3);
        fflush(fp4);

        /* plot graph */
        PlotOne();
        usleep(250); /* sleep for short time */
      }
}

void quit()
{  printf("\nctrl-c caught:\n Shutting down pipes\n");
   StopPlot();

   printf("closing data files\n");
   fclose(fp1);
   fclose(fp2);
   fclose(fp3);
   fclose(fp4);

   printf("deleting data files\n");
   RemoveDat();
}
```

The plotter.c module is as follows:

```
/* plotter.c module */
/* contains routines to plot a data file produced by another program  */
/* 2d data plotted in this version                                    */
/**********************************************************************/

#include "externals.h"

static FILE *plot1,
       *plot2,
       *ashell;
```

```
static char *startplot1 = "plot [] [0:1.1]'plot11.dat' with lines,
            'plot12.dat' with lines\n";

static char *startplot2 = "plot 'plot21.dat' with lines,
            'plot22.dat' with lines\n";

static char *replot = "replot\n";
static char *command1= "/usr/local/bin/gnuplot> dump1";
static char *command2= "/usr/local/bin/gnuplot> dump2";
static char *deletefiles = "rm plot11.dat plot12.dat plot21.dat plot22.dat";
static char *set_term = "set terminal x11\n";

void
StartPlot(void)
 { plot1 = popen(command1, "w");
   fprintf(plot1, "%s", set_term);
   fflush(plot1);
   if (plot1 == NULL)
      exit(2);
   plot2 = popen(command2, "w");
   fprintf(plot2, "%s", set_term);
   fflush(plot2);
   if (plot2 == NULL)
      exit(2);
 }

void
RemoveDat(void)
 { ashell = popen(deletefiles, "w");
   exit(0);
 }

void
StopPlot(void)
 { pclose(plot1);
   pclose(plot2);
 }

void
PlotOne(void)
 { fprintf(plot1, "%s", startplot1);
   fflush(plot1);

   fprintf(plot2, "%s", startplot2);
   fflush(plot2);
 }

void
RePlot(void)
 { fprintf(plot1, "%s", replot);
   fflush(plot1);
 }
```

The header file externals.h contains the following:

```
/* externals.h */
#ifndef EXTERNALS
#define EXTERNALS

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

/* prototypes */

void StartPlot(void);
void RemoveDat(void);
void StopPlot(void);
void PlotOne(void);
void RePlot(void);
#endif
```

Exercises

Exercise 12733

Setup a two-way pipe between parent and child processes in a C program. i.e. both can send and receive signals.

Dave Marshall
1/5/1999