Yacine Manseur

10/13/2015

Operating Systems

Problem Set 4

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 4

// catgrepmore.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <fcntl.h>

#include <sys/signal.h>

#include <sys/types.h>

#include <sys/wait.h>

int numFiles = 0, numBytes = 0;

void sig\_handler(int signo)

{

if (signo == SIGINT)

{

// Write total number of files and bytes processed.

fprintf(stderr, "Total files processed: %d\n", numFiles);

fprintf(stderr, "Total bytes processed: %d\n", numBytes);

exit(1);

}

else if (signo == SIGPIPE)

{

fprintf(stderr, "Broken pipe encountered.\n");

exit(1);

}

}

void processFile(char \*pattern, char \*infile)

{

int iFD, grepFD[2], moreFD[2];

int bufferSize = 4096;

int bytesRead = 0, bytesWrite = 0, bytesMissed = 0;

pid\_t pid1, pid2;

char \*buffer = malloc (bufferSize\*sizeof(char));

if(buffer == NULL)

{

fprintf(stderr, "Cannot allocate buffer with size: %d\n", bufferSize);

exit(1);

}

// Create the pipe

if( pipe(grepFD) == -1 || pipe(moreFD) == -1)

{

perror("Can't create pipe.");

exit(1);

}

switch (pid1 = fork())

{

case -1:

perror("Fork failed in grep.");

exit(1);

break;

case 0: // GREP

// Close dangling file descriptors

if( close(grepFD[1]) == -1 || close(moreFD[0]) == -1)

{

fprintf(stderr, "Cannot close input file %s: %s\n", infile, strerror(errno));

exit(1);

}

// Redirect grep to stdin and more to stdout

if( dup2(grepFD[0], 0) == -1 || dup2(moreFD[1], 1) == -1)

{

fprintf(stderr, "Cannot duplicate file descriptor in grep: %s\n", strerror(errno));

}

// Close redirected file descriptors

if( close(grepFD[0]) == -1 || close(moreFD[1]) == -1)

{

fprintf(stderr, "Cannot close input file %s: %s\n", infile, strerror(errno));

exit(1);

}

execlp("grep", "grep", pattern, NULL);

break;

default:

switch(pid2 = fork())

{

case -1:

perror("Fork failed in more.");

exit(1);

break;

case 0: // MORE

// Close dangling file descriptors

if( close(grepFD[0]) == -1 || close(grepFD[1]) == -1 || close(moreFD[1]) == -1)

{

fprintf(stderr, "Cannot close input file %s: %s\n", infile, strerror(errno));

exit(1);

}

// Redirect more to stdin

if (dup2(moreFD[0], 0) == -1)

{

fprintf(stderr, "Cannot duplicate file descriptor in more: %s\n", strerror(errno));

}

// Close redirected file descriptor

if (close(moreFD[0]) == -1){

fprintf(stderr, "Cannot close input file %s: %s\n", infile, strerror(errno));

exit(1);

}

execlp("more", "more", NULL);

break;

default: // PARENT

// Close dangling file descriptors

if( close(moreFD[0]) == -1 || close(moreFD[1]) == -1 || close(grepFD[0]) == -1)

{

fprintf(stderr, "Cannot close input file %s: %s\n", infile, strerror(errno));

exit(1);

}

iFD = open(infile, O\_RDONLY);

if (iFD == -1)

{

fprintf(stderr, "Cannot open input file: %s\n", infile);

exit(1);

}

while ((bytesRead = read(iFD, buffer, bufferSize)) > 0)

{

bytesWrite = write(grepFD[1], buffer, bytesRead);

if(bytesWrite < 0)

{

fprintf(stderr, "Write failed: %s\n", strerror(errno));

exit(1);

}

// Partial write should never occur

while (bytesWrite != bytesRead) //Partial write

{

bytesMissed = bytesRead - bytesWrite;

// write remaining bytes

bytesWrite += write(grepFD[1], buffer+bytesWrite, bytesMissed);

}

}

// Error when trying to read the input file

if (bytesRead < 0)

{

fprintf(stderr, "Cannot read file: %s\n", strerror(errno));

exit(1);

}

numBytes += bytesWrite;

// close infile and pipe

if( close(iFD) == -1 || close(grepFD[1]) == -1){

fprintf(stderr, "Cannot close input file %s: %s\n", infile, strerror(errno));

exit(1);

}

// Wait for state change

if(waitpid(pid2,NULL,0) == -1)

{

perror("Error waiting for state change.");

}

break;

}

// Wait for state change

if(waitpid(pid1, NULL, 0) == -1)

{

perror("Error waiting for state change.");

}

break;

}

free(buffer);

}

int main (int argc, char \*argv[])

{

if(signal(SIGINT, sig\_handler) == SIG\_ERR)

{

fprintf(stderr, "Cannot catch SIGINT: %s\n", strerror(errno));

}

if(signal (SIGPIPE, sig\_handler) == SIG\_ERR)

{

fprintf(stderr, "Cannot catch SIGPIPE: %s\n", strerror(errno));

}

int ii;

if(argc < 3)

{

fprintf(stderr, "Error: Incorrect syntax");

return -1;

}

for(ii = 2; ii < argc; ii++)

{

processFile(argv[1], argv[ii]);

numFiles++;

}

// Write total number of files and bytes processed.

fprintf(stderr, "Total files processed: %d\n", numFiles);

fprintf(stderr, "Total bytes processed: %d\n", numBytes);

exit(0);

}