ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2016-2017
ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ
ΜΑΘΗΜΑ: ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΟ 6 - ΣΗΜΕΙΩΣΕΙΣ

Σωληνώσεις (Pipes)

Οι **σωληνώσεις (pipes)** υιοθετούνται ώστε να συνδέσουν τη ροή δεδομένων από μια διεργασία σε κάποια άλλη. Γενικά, η έξοδος μιας διεργασίας συνδέεται με την είσοδο μιας άλλης διεργασίας όπως ακριβώς γίνεται και στην περίπτωση των εντολών του φλοιού. Σχηματικά, μια σωλήνωση έχει ως εξής:

Ο απλούστερος τρόπος για να περάσουμε δεδομένα από μια διεργασία σε μια άλλη είναι με τη βοήθεια των συναρτήσεων **popen** & **pclose**.

```
#include <stdio.h>

FILE *popen(const char *command, const char *open_mode);
int pclose(FILE *stream_to_close);
```

Η popen επιτρέπει ένα πρόγραμμα **να καλέσει ένα άλλο πρόγραμμα σαν μια νέα διεργασία** και να περάσει δεδομένα ή να λάβει δεδομένα προς / από αυτή. Το όνομα του προγράμματος που θα εκτελεστεί μπαίνει ως παράμετρος στη συνάρτηση μαζί με μια παράμετρο open_mode που μπορεί να είναι είτε r ή w. Όταν είναι r, το αποτέλεσμα του καλούμενου προγράμματος είναι διαθέσιμο στη διεργασία που εκτελεί την κλήση μέσω του αρχείου που επιστρέφει η popen. Αν είναι w το πρόγραμμα μπορεί να στέλνει δεδομένα μέσω κλήσεων της fwrite. Έπειτα το πρόγραμμα κλήσης μπορεί να διαβάσει τα δεδομένα μέσω της προκαθορισμένης εισόδου. Επειδή με την κλήση της popen η δεύτερη παράμετρος μπορεί να είναι μόνο r ή w, αν επιθυμούμε αμφίδρομη επικοινωνία, τότε πρέπει να υιοθετήσουμε δύο σωληνώσεις, μια για κάθε κατεύθυνση.

Η pclose επιτελεί το κλείσιμο του αρχείου έπειτα από την επεξεργασία των δεδομένων που στέλνουμε προς τις διεργασίες. Η pclose θα επιστρέψει μόνο όταν η διεργασία που κλήθηκε με την popen τελειώσει.

Code 1

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

int main()
{
        FILE *read_fp;
        char buffer[BUFSIZ + 1];
        int chars_read;
        memset(buffer, '\0', sizeof(buffer));
        read_fp = popen("uname -a", "r");
```

```
        if (read_fp != NULL) {
                chars_read = fread(buffer, sizeof(char), BUFSIZ, read_fp);
                if (chars_read > 0) {
                        printf("Output was:-\n%s\n", buffer);
                }
                pclose(read_fp);
                exit(EXIT_SUCCESS);
        }
        exit(EXIT_FAILURE);
}
```

Στο ακόλουθο παράδειγμα, στέλνουμε κάποια δεδομένα στο πρόγραμμα το οποίο εκτελούμε από τη διεργασία μας.

Code 2

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

int main()
{
        FILE *write_fp;
        char buffer[BUFSIZ + 1];
        sprintf(buffer, "Once upon a time, there was...\n");
        write_fp = popen("od -c", "w");
        if (write_fp != NULL) {
                fwrite(buffer, sizeof(char), strlen(buffer), write_fp);
                pclose(write_fp);
                exit(EXIT_SUCCESS);
        }
        exit(EXIT_FAILURE);
}
```

Το επόμενο παράδειγμα επιτρέπει την ανάγνωση δεδομένων από μια διεργασία που καλύπτουν όλο το φάσμα των αποτελεσμάτων της.

Code 3

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main()
{
        FILE *read_fp;
        char buffer[BUFSIZ + 1];
        int chars_read;
        memset(buffer, '\0', sizeof(buffer));
```

```
        read_fp = popen("ps ax", "r");
        if (read_fp != NULL) {
                chars_read = fread(buffer, sizeof(char), BUFSIZ, read_fp);
                while (chars_read > 0) {
                        buffer[chars_read – 1] = '\0';
                        printf("Reading %d:-\n %s\n", BUFSIZ, buffer);
                        chars_read = fread(buffer, sizeof(char), BUFSIZ, read_fp);
                }
                pclose(read_fp);
                exit(EXIT_SUCCESS);
        }
        exit(EXIT_FAILURE);
}
```

Σε πιο κάτω επίπεδο, για την επικοινωνία διεργασιών, διατίθεται η συνάρτηση **pipe**. Προσφέρει ένα τρόπο για το πέρασμα δεδομένων ανάμεσα σε προγράμματα χωρίς την ανάγκη για επιπλέον δεδομένα που απαιτούνται για την κλήση του φλοιού και τη διερμηνεία της ζητούμενης εντολής.

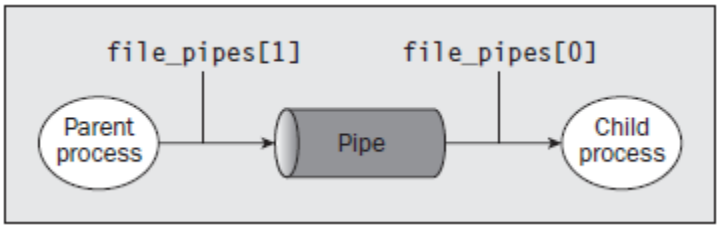
```
#include <unistd.h>

int pipe(int file_descriptor[2]);
```

Η συνάρτηση παίρνει ως παράμετρο ένα πίνακα με δύο ακεραίους περιγραφείς αρχείων. Οι δύο περιγραφείς αρχείων συνδέονται ως εξής: ότι δεδομένα καταγράφονται στο file_descriptor[1] μπορούν να διαβαστούν μέσω του file_descriptor[0]. Τα δεδομένα επεξεργάζονται με ένα FIFO μοντέλο.

Code 4

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main()
{
        int data_processed;
        int file_pipes[2];
        const char some_data[] = "123";
        char buffer[BUFSIZ + 1];
        memset(buffer, '\0', sizeof(buffer));
        if (pipe(file_pipes) == 0) {
                data_processed = write(file_pipes[1], some_data, strlen(some_data));
                printf("Wrote %d bytes\n", data_processed);
                data_processed = read(file_pipes[0], buffer, BUFSIZ);
                printf("Read %d bytes: %s\n", data_processed, buffer);
                exit(EXIT_SUCCESS);
        }
        exit(EXIT_FAILURE);
}
```

Το ακόλουθο παράδειγμα παρουσιάζει την επικοινωνία μιας πατρικής και μιας διεργασίας παιδί με τη βοήθεια σωληνώσεων. Το ακόλουθο σχήμα παρουσιάζει την επικοινωνία αυτή:



Code 5

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main()
{
int data_processed;
	int file_pipes[2];
	const char some_data[] = "123";
	char buffer[BUFSIZ + 1];
	pid_t fork_result;
	memset(buffer, '\0', sizeof(buffer));
	if (pipe(file_pipes) == 0) {
		fork_result = fork();
		if (fork_result == -1) {
			fprintf(stderr, "Fork failure");
			exit(EXIT_FAILURE);
		}
		if (fork_result == 0) {
			data_processed = read(file_pipes[0], buffer, BUFSIZ);
			printf("Read %d bytes: %s\n", data_processed, buffer);
			exit(EXIT_SUCCESS);
		}
		else {
			data_processed = write(file_pipes[1], some_data,
			strlen(some_data));
			printf("Wrote %d bytes\n", data_processed);
		}
	}
	exit(EXIT_SUCCESS);
}
```

Ακολουθεί ένα παράδειγμα στο οποίο η πατρική διεργασία και η διεργασία παιδί είναι διαφορετικές εντελώς διεργασίες. Η υλοποίηση υιοθετεί την εντολή exec.

Code 6 - Parent

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

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main()
{
	int data_processed;
	int file_pipes[2];
	const char some_data[] = "123";
	char buffer[BUFSIZ + 1];
	pid_t fork_result;
	memset(buffer, '\0', sizeof(buffer));
	if (pipe(file_pipes) == 0) {
		fork_result = fork();
		if (fork_result == (pid_t)-1) {
			fprintf(stderr, "Fork failure");
			exit(EXIT_FAILURE);
		}
		if (fork_result == 0) {
			sprintf(buffer, "%d", file_pipes[0]);
			(void)execl("Code6_Child", "Code6_Child", buffer, (char *)0);
			exit(EXIT_FAILURE);
		}
		else {
			data_processed = write(file_pipes[1], some_data,
			strlen(some_data));
			printf("%d - wrote %d bytes\n", getpid(), data_processed);
		}
	}
	exit(EXIT_SUCCESS);
}
```

Code 6 - Child

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
	int data_processed;
	char buffer[BUFSIZ + 1];
	int file_descriptor;
	memset(buffer, '\0', sizeof(buffer));
	sscanf(argv[1], "%d", &file_descriptor);
	data_processed = read(file_descriptor, buffer, BUFSIZ);
	printf("%d - read %d bytes: %s\n", getpid(), data_processed, buffer);
	exit(EXIT_SUCCESS);
}
```

Named Pipes - FIFOs

Στα παραπάνω παραδείγματα, έχουμε δει το πέρασμα δεδομένων μεταξύ διεργασιών που είχαν κοινό πρόγονο. Είναι όμως γεγονός πως τέτοιου είδους περιπτώσεις δεν είναι αποδοτικές όταν θέλουμε άσχετες διεργασίες να ανταλλάξουν δεδομένα. Σε αυτές τις περιπτώσεις χρησιμοποιούμε τις λεγόμενες **named pipes – FIFOs**. Μια named pipe είναι ένα αρχείο ειδικού τύπου (όλα στο Linux είναι αρχεία) που υπάρχει στο δίσκο αλλά συμπεριφέρεται σαν τις σωληνώσεις που είδαμε πιο πάνω. Οι συναρτήσεις που προσφέρονται για τη διαχείριση των FIFOs είναι οι ακόλουθες:

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *filename, mode_t mode);
int mknod(const char *filename, mode_t mode | S_IFIFO, (dev_t) 0);
```

Η δημιουργία μιας FIFO μπορεί να γίνει με τον ακόλουθο κώδικα:

Code 7

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

int main()
{
        int res = mkfifo("/tmp/my_fifo", 0777);
        if (res == 0) printf("FIFO created\n");
                exit(EXIT_SUCCESS);
}
```

Μπορούμε να διαχειριστούμε το FIFO αρχείο κανονικά όπως ένα οποιοδήποτε άλλο αρχείο. Για το άνοιγμα των FIFOs μπορούμε να βασιστούμε στη συνάρτηση open. Η συνάρτηση παίρνει ως παραμέτρους τη διαδρομή προς το συνδεδεμένο αρχείο καθώς και το αν η σωλήνωση θα ανοίξει για ανάγνωση ή εγγραφή. Οι συνδυασμοί των παραμέτρων της open είναι οι ακόλουθοι:

- open(const char *path, O_RDONLY). Η συνάρτηση δεν θα επιστρέψει παρά μόνο αν το ίδιο αρχείο ανοίξει από μια άλλη διεργασία για εγγραφή.
- open(const char *path, O_RDONLY | O_NONBLOCK). Η συνάρτηση θα επιστρέψει άμεσα ασχέτως αν το ίδιο αρχείο ανοίξει από μια άλλη διεργασία για εγγραφή.
- open(const char *path, O_WRONLY). Η συνάρτηση δεν θα επιστρέψει παρά μόνο αν το ίδιο αρχείο ανοίξει από μια άλλη διεργασία για ανάγνωση.
- open(const char *path, O_WRONLY | O_NONBLOCK). Η συνάρτηση θα επιστρέψει άμεσα ασχέτως αν το ίδιο αρχείο ανοίξει από μια άλλη διεργασία για ανάγνωση.

Code 8 - A

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

```
#include <fcntl.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FIFO_NAME "/tmp/my_fifo"
#define BUFFER_SIZE PIPE_BUF
#define TEN_MEG (1024 * 1024 * 10)

int main()
{
	int pipe_fd;
	int res;
	int open_mode = O_WRONLY;
	int bytes_sent = 0;
	char buffer[BUFFER_SIZE + 1];
	if (access(FIFO_NAME, F_OK) == -1) {
		res = mkfifo(FIFO_NAME, 0777);
		if (res != 0) {
			fprintf(stderr, "Could not create fifo %s\n", FIFO_NAME);
			exit(EXIT_FAILURE);
		}
	}
	printf("Process %d opening FIFO O_WRONLY\n", getpid());
	pipe_fd = open(FIFO_NAME, open_mode);
	printf("Process %d result %d\n", getpid(), pipe_fd);
	if (pipe_fd != -1) {
		while(bytes_sent < TEN_MEG) {
			res = write(pipe_fd, buffer, BUFFER_SIZE);
			if (res == -1) {
				fprintf(stderr, "Write error on pipe\n");
				exit(EXIT_FAILURE);
			}
			bytes_sent += res;
		}
		(void)close(pipe_fd);
	}
	else {
		exit(EXIT_FAILURE);
	}
	printf("Process %d finished\n", getpid());
	exit(EXIT_SUCCESS);
}
```

Code 8 - B

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

```
#include <string.h>
#include <fcntl.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FIFO_NAME "/tmp/my_fifo"
#define BUFFER_SIZE PIPE_BUF

int main()
{
        int pipe_fd;
        int res;
        int open_mode = O_RDONLY;
        char buffer[BUFFER_SIZE + 1];
        int bytes_read = 0;
        memset(buffer, '\0', sizeof(buffer));
        printf("Process %d opening FIFO O_RDONLY\n", getpid());
        pipe_fd = open(FIFO_NAME, open_mode);
        printf("Process %d result %d\n", getpid(), pipe_fd);
        if (pipe_fd != -1) {
                do {
                        res = read(pipe_fd, buffer, BUFFER_SIZE);
                        bytes_read += res;
                } while (res > 0);
                (void)close(pipe_fd);
        }
        else {
                exit(EXIT_FAILURE);
        }
        printf("Process %d finished, %d bytes read\n", getpid(), bytes_read);
        exit(EXIT_SUCCESS);
}
```

Το επόμενο παράδειγμα μας δείχνει την επικοινωνία ανάμεσα σε ένα εξυπηρετητή και ένα πελάτη με τη βοήθεια των FIFOs.

Code 9 – Client.h

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>

#define SERVER_FIFO_NAME "/tmp/serv_fifo"
```

```
#define CLIENT_FIFO_NAME "/tmp/cli_%d_fifo"
#define BUFFER_SIZE 20

struct data_to_pass_st {
	pid_t client_pid;
	char some_data[BUFFER_SIZE - 1];
};
```

Code 9 – Server

```
#include "client.h"
#include <ctype.h>

int main()
{
	int server_fifo_fd, client_fifo_fd;
	struct data_to_pass_st my_data;
	int read_res;
	char client_fifo[256];
	char *tmp_char_ptr;
	mkfifo(SERVER_FIFO_NAME, 0777);
	server_fifo_fd = open(SERVER_FIFO_NAME, O_RDONLY);
	if (server_fifo_fd == -1) {
		fprintf(stderr, "Server fifo failure\n");
		exit(EXIT_FAILURE);
	}
	sleep(10); /* lets clients queue for demo purposes */
	do {
		read_res = read(server_fifo_fd, &my_data, sizeof(my_data));
		if (read_res > 0) {
			tmp_char_ptr = my_data.some_data;
			while (*tmp_char_ptr) {
				*tmp_char_ptr = toupper(*tmp_char_ptr);
				tmp_char_ptr++;
			}
			sprintf(client_fifo, CLIENT_FIFO_NAME, my_data.client_pid);
			client_fifo_fd = open(client_fifo, O_WRONLY);
			if (client_fifo_fd != -1) {
				write(client_fifo_fd, &my_data, sizeof(my_data));
				close(client_fifo_fd);
			}
		}
	} while (read_res > 0);
	close(server_fifo_fd);
	unlink(SERVER_FIFO_NAME);
	exit(EXIT_SUCCESS);
}
```

Code 9 – Client

```
#include "client.h"
#include <ctype.h>

int main()
{
	int server_fifo_fd, client_fifo_fd;
	struct data_to_pass_st my_data;
	int times_to_send;
	char client_fifo[256];
	server_fifo_fd = open(SERVER_FIFO_NAME, O_WRONLY);
	if (server_fifo_fd == -1) {
		fprintf(stderr, "Sorry, no server\n");
		exit(EXIT_FAILURE);
	}
	my_data.client_pid = getpid();
	sprintf(client_fifo, CLIENT_FIFO_NAME, my_data.client_pid);
	if (mkfifo(client_fifo, 0777) == -1) {
		fprintf(stderr, "Sorry, can't make %s\n", client_fifo);
		exit(EXIT_FAILURE);
	}
	for (times_to_send = 0; times_to_send < 5; times_to_send++) {
		sprintf(my_data.some_data, "Hello from %d", my_data.client_pid);
		printf("%d sent %s, ", my_data.client_pid, my_data.some_data);
		write(server_fifo_fd, &my_data, sizeof(my_data));
		client_fifo_fd = open(client_fifo, O_RDONLY);
		if (client_fifo_fd != -1) {
			if (read(client_fifo_fd, &my_data, sizeof(my_data)) > 0) {
				printf("received: %s\n", my_data.some_data);
			}
			close(client_fifo_fd);
		}
	}
	close(server_fifo_fd);
	unlink(client_fifo);
	exit(EXIT_SUCCESS);
}
```