با نام خدا

گزارش کار ازمایشگاه سیستم عامل

گزارش شماره 7

pipe

امیر حسین متقیان ۴۰۱۳۱۰۴۳

کیان پور اذر ۴۰۱۳۱۴۰۳

در این ازمایش در مورد pipe که یکی از روش های انتقال اطلاعات بین فرایند ها (ICP) است صحبت میکنیم

در این ازمایش از Unidirectional pipe ها استفاده میکنیم به این صورت که از یک طرف فقط read میکنیم و از طرف دیگر فقط write میکنیم

در این ازمایش باید با فرایند پدر یک پیام برای فرایند فرزند از طریق pipe بفرستیم و فرایند فرزند حروف کوچک پیام را بزرک و حروف بزرگ را کوچک کند و از طریق pipe برای والد بفرستند.از انجا که pipe های ما یک طرفه اند برای ان کار به دو pipe نیاز داریم یکی برای نوشتن در والد و خواندن در فرزند و دیگری برای نوشتن در فرزند و خواندن در والد

کد ساخت pipe

```
// Create pipes and declare buffer
int parent_fd[2];
int child_fd[2];
char buffer[100];
char message[] = "This Is The First Process\n";

// Initialize the parent pipe
if (pipe(parent_fd) == -1) {
   perror("parent pipe creation failed");
   exit(EXIT_FAILURE);
}

// Initialize the child pipe
if (pipe(child_fd) == -1) {
   perror("child pipe creation failed");
   exit(EXIT_FAILURE);
}
```

ارایه بافر برای خواندن از pipe مورد استفاده قرار میگیرد

از تابع toggleCase برای تبدیل حروف استفاده میکنیم

```
void toggleCase(char *str) {
   while (*str) {
      if (islower(*str)) {
         *str = toupper(*str);
      } else if (isupper(*str)) {
         *str = tolower(*str);
      }
      str++;
   }
}
```

فرایند والد

```
// Parent process
else {
   // Close unused read end of the parent pipe
   close(parent_fd[0]);
   // Close unused write end of the child pipe
   close(child_fd[1]);
   printf("Parent process: created\n");

   // Write to parent pipe
   if (write(parent_fd[1], message, sizeof(message)) == -1) {
      perror("write failed");
      exit(EXIT_FAILURE);
   }
   printf("Parent process: sent value\n");

   // Read from child pipe
   ssize_t n = read(child_fd[0], buffer, sizeof(buffer));
   if (n == -1) {
      perror("read failed");
      exit(EXIT_FAILURE);
   }

   // Null-terminate the string
   buffer[n] = '\0';
   printf("Parent process: received modified message: %s", buffer);
```

```
    // Close the pipes
    close(parent_fd[1]);
    close(child_fd[0]);

    // Wait for child process to finish
    wait(nullptr);
    printf("Parent process finished\n");
}
```

در اینجا پس از بستن سر هایی از pipe که از انها استفاه نمیکنیم پیام را در یکی از pipe ها نوشته و سپس از pipe دیگر پیام تغییر یافته را میخوانیم

فرایند فرزند

```
// Close unused write end of the parent pipe
close(parent_fd[1]);
// Close unused read end of the child pipe
close(child_fd[0]);
printf("Child process: created\n");

// Read from parent pipe
const ssize_t n = read(parent_fd[0], buffer, sizeof(buffer));
if (n == -1) {
    perror("read failed");
    exit(EXIT_FAILURE);
}

// Null-terminate the string
buffer[n] = '\0';

// Write the received message to stdout
printf("Child process: received message: %s", buffer);

// Convert message
toggleCase(buffer);

// Write to child pipe
if (write(child_fd[1], buffer, n) == -1) {
    perror("write failed");
    exit(EXIT_FAILURE);
}
printf("Child process: sent modified value\n");
```

```
// Close pipes
close(parent_fd[0]);
close(child_fd[1]);
printf("Child process: finished\n");
```

در اینجا نیز پس از بستن سر هایی از pipe که از انها استفاده نمیکنیم پیغام را از یکی از pipe ها میخوانیم سپس ان را با تابع toggleCase تغییر میدهیم و در pipe دیگر برای والد ارسال میکنیم

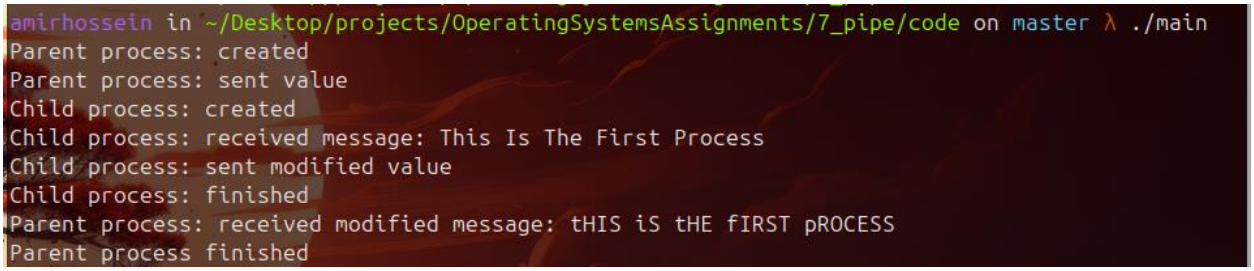
تمام کد

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

void toggleCase(char *str) {
    while (*str) {
        if (islower(*str)) {
            *str = toupper(*str);
        } else if (isupper(*str)) {
            *str = tolower(*str);
        }
        str++;
    }
}

int main(void) {
    // Create pipes and declare buffer
    int parent_fd[2];
    int child_fd[2];
    char buffer[100];
    char message[] = "This Is The First Process\n";

    // Initialize the parent pipe
    if (pipe(parent_fd) == -1) {
        perror("parent pipe creation failed");
        exit(EXIT_FAILURE);
    }
```

```
// Initialize the child pipe
if (pipe(child_fd) == -1) {
    perror("child pipe creation failed");
    exit(EXIT_FAILURE);
}

// Fork the main process
const pid_t pid = fork();

// Failed to fork
if (pid < 0) {
    perror("failed to fork process");
    exit(EXIT_FAILURE);
}
// Child process
else if (pid == 0) {
    // Close unused write end of the parent pipe
    close(parent_fd[1]);
    // Close unused read end of the child pipe
    close(child_fd[0]);
    printf("Child process: created\n");

    // Read from parent pipe
    const ssize_t n = read(parent_fd[0], buffer, sizeof(buffer));
    if (n == -1) {
        perror("read failed");
        exit(EXIT_FAILURE);
    }

    // Null-terminate the string
    buffer[n] = '\0';

    // Write the received message to stdout
    printf("Child process: received message: %s", buffer);

    // Convert message
    toggleCase(buffer);

    // Write to child pipe
    if (write(child_fd[1], buffer, n) == -1) {
        perror("write failed");
        exit(EXIT_FAILURE);
    }
    printf("Child process: sent modified value\n");
```

```
    // Close pipes
    close(parent_fd[0]);
    close(child_fd[1]);
    printf("Child process: finished\n");
  }
  // Parent process
  else {
    // Close unused read end of the parent pipe
    close(parent_fd[0]);
    // Close unused write end of the child pipe
    close(child_fd[1]);
    printf("Parent process: created\n");

    // Write to parent pipe
    if (write(parent_fd[1], message, sizeof(message)) == -1) {
      perror("write failed");
      exit(EXIT_FAILURE);
    }
    printf("Parent process: sent value\n");

    // Read from child pipe
    ssize_t n = read(child_fd[0], buffer, sizeof(buffer));
    if (n == -1) {
      perror("read failed");
      exit(EXIT_FAILURE);
    }

    // Null-terminate the string
    buffer[n] = '\0';
    printf("Parent process: received modified message: %s", buffer);

    // Close the pipes
    close(parent_fd[1]);
    close(child_fd[0]);

    // Wait for child process to finish
    wait(nullptr);
    printf("Parent process finished\n");
  }

  return 0;
}
```

خروجی کد


amirhossein in ~/Desktop/projects/OperatingSystemsAssignments/7_pipe/code on master λ ./main
Parent process: created
Parent process: sent value
Child process: created
Child process: received message: This Is The First Process
Child process: sent modified value
Child process: finished
Parent process: received modified message: tHIS iS tHE fIRST pROCESS
Parent process finished