Betriebssysteme

Labor 11

Assignment 8

- Deadline: Woche 13
- Kommentare werden im Code benötigt!! !!
- Die Aufgaben sind sowohl auf Teams als auch auf Moodle geladen.

- **Achtung!** Bei den Aufgaben, bei denen die Begriffe "Client" und "Server" auftreten, wird berücksichtigt, dass es sich bei beiden um Prozesse handelt und die Kommunikation über Pipes erfolgt.

  Für Labor 10 müssen alle Aufgaben mit benannten Pipes (FIFO) gelöst werden.

Kommunikation zwischen Unix-Prozesse: FIFO

- **Externe Kanäle**

- Benannten Pipes wurden seit der `UNIX SYSTEM III`−Version eingeführt und sind eine Kombination aus regulären Dateien und Pipes. Sie haben einen symbolischen Namen, der wie jede normale Datei in einem Verzeichnis erstellt wird, behalten jedoch alle Eigenschaften von Pipe-Dateien bei. Benannten Pipe-Dateien werden auch als **FIFO**-Dateien bezeichnet.

Kommunikation zwischen Unix-Prozesse: FIFO

- Der Vorteil von benannten Pipe-Dateien gegenüber namenlosen Pipe-Dateien besteht darin, dass die Prozesse, die miteinander kommunizieren, ihre Deskriptoren nicht senden müssen. Infolgedessen kann die Kommunikation zwischen Prozessen erfolgen, die nicht von dem Prozess stammen dürfen, der die Datei geöffnet hat, wie dies bei namenlosen Pipe-Dateien der Fall ist. In diesem Fall kann jeder Prozess beim Lesen oder Schreiben seiner Pipe-Datei mit Namen geöffnet werden, wobei der symbolische Name wichtig ist.
- **`Beobachtung`**:
  - Eine FIFO-Datei wird mit dem UNIX-Befehl `ls -l` angezeigt, der den Buchstaben `p` im ersten Feld der Zugriffsrechte anzeigt.

Kommunikation zwischen Unix-Prozesse: FIFO

Erzeugung von FIFO-Dateien

In der SVR3-Version wurde die Erzeugung einer FIFO-Datei durch Aufrufen der Systemfunktion `mknod` durchgeführt. Die Schnittstelle dieser Funktion ist:

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *nume, mode_t tip_perm_acces)
```

wobei:

- `nume` steht für die FIFO-Datei;
- `tip_perm_acces` ist eine ganze Zahl, die den Typ der `S_FIFO`-Datei (bei benannten Pipes) und die Zugriffsrechte (r, w, x für den Benutzer, die Gruppe und andere Benutzer) angibt. Der Parameter hat folgende Form: `S_FIFO | Rechte`.

Kommunikation zwischen Unix-Prozesse: FIFO

Öffnen einer FIFO-Datei

Das Öffnen einer FIFO-Datei beim Lesen oder Schreiben erfolgt wie bei jeder normalen Datei mit der `open`-Funktion.

Lesen aus einer FIFO-Datei

Das Lesen aus FIFO-Dateien erfolgt mit der `read`-Systemfunktion, die wie bei jeder normalen Datei verwendet wird.

Kommunikation zwischen Unix-Prozesse: FIFO

Schreiben in eine FIFO-Datei

Das Schreiben in eine FIFO-Datei erfolgt mit der `write`-Systemfunktion, die wie bei jeder normalen Datei verwendet wird. Wenn das Kennzeichen `O_NDELAY` oder `O_NONBLOCK` beim Öffnen gesetzt ist und die FIFO-Datei voll ist oder nicht genügend Speicherplatz zum vollständigen Schreiben vorhanden ist, wird der Prozess nicht blockiert, bis die Daten von einem anderen Prozess gelesen wurden, aber die `write`-Funktion gibt **Null** zurück.

Wenn Sie versuchen, in eine FIFO-Datei zu schreiben, an die kein Prozess zum Lesen angehängt ist, wird das `SIGPIPE`-Signal generiert.

Kommunikation zwischen Unix-Prozesse: FIFO

Wenn mehrere Prozesse in eine FIFO-Datei schreiben, wird das Schreiben als atomare Operation betrachtet, solange das durch eine Schreiboperation geschriebene Datenvolumen die durch die symbolische Konstante `PIPE_BUF` angegebene Höchstgrenze nicht überschreitet. Andernfalls können Schreibvorgänge mit mehreren Prozessen gemischt werden.

Kommunikation zwischen Unix-Prozesse: FIFO

Schließen einer FIFO-Datei

Das Schließen der FIFO-Datei erfolgt mit der `close`-Systemfunktion, die wie bei jeder normalen Datei verwendet wird. Das Schließen einer FIFO-Datei umfasst das Beenden der Datei für den Prozess, der aus der FIFO-Datei geliest.

Löschen einer FIFO-Datei

Im Gegensatz zu Pipe-Dateien erfolgt das Löschen von FIFO-Dateien durch Aufrufen der `unlink`-Funktion. Die Datei gilt als gelöscht, wenn die Anzahl der Links Null wird.

Kommunikation zwischen Unix-Prozesse: FIFO

Die Kommunikation zwischen Prozessen über FIFO-Dateien erfolgt wie folgt:

1. Ein Prozess erstellt die FIFO-Datei unter Angabe ihres symbolischen Namens und ruft die `mknod`- oder `mkfifo`-Systemfunktion auf.
2. Der Prozess, der Daten an andere übermittelt, öffnet die Datei mit der `open`-Systemfunktion und schreibt diese Daten mit der `write`-Funktion.
3. Der Prozess, der Daten empfängt, öffnet die FIFO-Datei beim Lesen mit der `open`-Systemfunktion und liest dann die Daten mit der `read`-Funktion daraus.

`Beobachtung:`

Die Kommunikation erfolgt in eine Richtung oder in beide Richtungen!

Beispielprogramme

1. Schreiben Sie 4 verschiedene Programme für: Erstellen, Löschen, Schreiben und Lesen aus einem Fifo.

Beispielprogramme

header.h

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

#define FISNAME "fifo1234"

extern int errno;
```

Beispielprogramme

erzeugung.c

```
#include "header.h"

int main(){
  if((mkfifo(FISNAME,
0666))&&(errno!=EEXIST)){
    printf("FIFO-Fehler\n");
    exit(0);
  }
  printf("Ein Fifo wurde erstellt\n");
}
```

loeschen.c

```
#include "header.h"

int main(){
  if(unlink(FISNAME) < 0){
    printf("Fehler beim Löschen von
FIFO\n");
    exit(0);
  }
  printf("FIFO wurde gelöscht\n");
}
```

Beispielprogramme

schreiben.c

```
#include <string.h>
#include "header.h"

int main(){
  int fd, n;
  char s[20];

  fd=open(FISNAME, O_WRONLY); // es ist selbsthemmend. Wartet bis ein anderer Prozess es mit O_RDONLY öffnet
                              //man kann O_RDWR setzen, um dies zu vermeiden
  if( fd < 0){
    printf("FIFO-Öffnungsfehler\n");
    exit(1);
  }
  strcpy(s, "Wir schreiben in FIFO");
  n=strlen(s);
  if(write(fd,s,n) != n){
    printf("FIFO-Schreibfehler\n");
    exit(1);
  }
  printf("Es wurden in FIFO %d Bytes geschrieben\n", n);
  close(fd);
}
```

Beispielprogramme

lesen.c

```
#include <string.h>
#include "header.h"

int main(){
  int fd, n;
  char s[20];

  fd=open(FISNAME, O_RDONLY);
  if( fd < 0){
    printf("FIFO-Öffnungsfehler\n");
    exit(1);
  }
  if((n=write(fd,s,14)) < 0){
    printf("Fehler beim Lesen aus der FIFO\n");
    exit(1);
  }
  s[n+1]='\0';
  printf("Es wurden aus der FIFO %d Bytes gelesen: %s\n", n, s);
  close(fd);
}
```

Kommunikation zwischen Unix-Prozesse: FIFO

Was bei einem FIFO zu beachten ist:

Bevor Sie ein Client-Server-Beispiel programmieren werden, müssen Sie noch einiges wissen, worauf Sie beim Zugriff auf FIFOs achten müssen:

- Wenn Sie beim Öffnen des FIFO mit `open()` nicht den Modus `O_NONBLOCK` verwenden, wird die Öffnung des FIFO sowohl zum Schreiben als auch zum Lesen blockiert (nicht bei `O_RDWR`). Beim Schreiben wird so lange blockiert, bis ein anderer Prozess das FIFO zum Lesen geöffnet hat. Beim Lesen wiederum wird ebenfalls so lange blockiert, bis ein anderer Prozess in das FIFO schreibt.
- Das Flag `O_NONBLOCK` kann nur bei Lesezugriffen verwendet werden. Wird mit `O_NONBLOCK` versucht, das FIFO mit Schreibzugriff zu öffnen, führt dies zu einem Fehler beim Öffnen.
- Wenn Sie ein FIFO zum Schreiben mit `close()` oder `fclose()` schließen, bedeutet dies für das FIFO zum Lesen ein EOF.
- Wie schon bei den Pipes gilt, falls mehrere Prozesse auf dasselbe FIFO schreiben, muss darauf geachtet werden, dass niemals mehr als `PIPE_BUF` Bytes auf einmal geschrieben werden. Dies daher, damit die Daten nicht durcheinander gemischt werden.

Beispielprogramme

2. Die Anzahl der Bytes und gleichzeitig geöffneter FIFOs können Sie mit dem folgenden Listing ermitteln.

Beispielprogramme

```
/* fifo_buf.c */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>
int main (void) {
   /*Wir erzeugen das FIFO*/
   if ((mkfifo ("fifo0001",  S_IRUSR |
S_IWUSR)) == -1) {
      /* FIFO bereits vorhanden - kein
fataler Fehler */
      if(errno == EEXIST)
         perror ("mkfifo()");
```

```
else {
         perror("mkfifio()");
         exit (EXIT_FAILURE);
      }
   }
   printf("Es können max. %ld Bytes in das
FIFO geschrieben"
          "werden\n", pathconf ("fifo0001",
_PC_PIPE_BUF));
   printf ("Außerdem können max %ld FIFOs
geöffnet sein\n",
            sysconf (_SC_OPEN_MAX));
   return EXIT_SUCCESS;
}
```

Beispielprogramme

3. Wie können sich Eltern- und Kindprozess miteinander über ein FIFO unterhalten. Der Kindprozess soll etwas in das FIFO schreiben, was der Elternprozess dann ausliest. Ein einfaches Server-Client-Beispiel (Eltern/Kind).

Beispielprogramme

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <errno.h>
#define FIFO "/tmp/fifo0001.1"
int main (void) {
   int r_fifo, w_fifo;
   char puffer[] =
      "Der Text für das FIFO an den Elternprozess\n";
   char buf[100];
   pid_t pid;
   if ((mkfifo (FIFO, S_IRUSR | S_IWUSR)) == -1) {
      /* FIFO bereits vorhanden - kein fataler Fehler */
      if(errno == EEXIST)
         perror ("mkfifo()");
      else {
         perror("mkfifo()");
         exit (EXIT_FAILURE);
      }
   }
   pid = fork ();
   if (pid == -1) {
      perror ("fork()");
      exit (EXIT_FAILURE);
   }
```

```
 else if (pid > 0) { /*Elternprozess liest aus dem FIFO */
      if ((r_fifo = open (FIFO, O_RDONLY)) < 0) {
         perror ("open()");
         exit (EXIT_FAILURE);
      }
      /*Wir warten auf das Ende vom Kindprozess */
      while (wait (NULL) != pid);
      /*Lesen aus dem FIFO */
      read (r_fifo, buf, strlen (puffer));
      buf[strlen(puffer)] = '\0';
      printf(" --- Elternprozess ---\n");
      printf ("%s", buf);
      close (r_fifo);
   }
   else {       /*Kindprozess schreibt in das FIFO */
      printf(" --- Kindprozess ---\n");
      if ((w_fifo = open (FIFO, O_WRONLY)) < 0) {
         perror ("open()");
         exit (EXIT_FAILURE);
      }
      /*Schreiben in das FIFO */
      write (w_fifo, puffer, strlen (puffer));
      close (w_fifo);   /* EOF */
      exit (EXIT_SUCCESS);
   }
   printf(" --- Ende ---\n");
   return EXIT_SUCCESS;
}
```

Beispielprogramme

4. Echtes Client-Server-Beispiel. Client sendet eine Nachricht und Server antwortet dem Sender.

Beispielprogramme

```
/* client.c */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define BUF 4096
int main (void) {
   char puffer[BUF], inhalt[BUF], delete_fifo[BUF];
   int fd, fdx;
   sprintf (puffer, "fifo.%d", getpid ());
   inhalt[0] = '\0';
   /*Erste Zeile der Nachricht enthält die PID */
   sprintf (inhalt, "%d\n", getpid ());
   /*Alle Zugriffsrechte der Dateikreierungsmaske
erlauben*/
   umask(0);
   if (mkfifo (puffer, O_RDWR | 0666) < 0) {
      /* FIFO bereits vorhanden - kein fataler Fehler */
      if(errno == EEXIST)
         printf ("Versuche; vorh. FIFO zu verwenden\n");
      else {
         perror("mkfifio()");
         exit (EXIT_FAILURE);}}
```

```
 fd = open ("fifo1.1", O_WRONLY);
   fdx = open (puffer, O_RDWR);
   if (fd == -1 || fdx == -1) {
      perror ("open()");
      exit (EXIT_FAILURE);
   }
   strcmp (delete_fifo, puffer);
   printf ("Bitte geben Sie Ihre Nachricht ein"
           " (Mit STRG+D beenden)\n>");
   while (fgets (puffer, BUF, stdin) != NULL) {
      if( strlen(inhalt) + strlen(puffer) < BUF ) {
            strcat (inhalt, puffer);
            printf (">");
      }
      else
            break;
   }
   inhalt[BUF] = '\0';
   write (fd, inhalt, BUF);
   if (read (fdx, puffer, BUF))
      printf ("%s\n", puffer);
   /*Antwort-FIFO wieder löschen */
   unlink (delete_fifo);
   return EXIT_SUCCESS;
}
```

Beispielprogramme

```
/* server.c */
#include <stdio.h>
#include <unistd. h.>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define BUF 4096
#define TEXT "Habe Ihre Anfrage soeben erhalten\n"
static void an_den_drucker (const char *text) {
   FILE *p;
   /*Pipe zum Tool lpr erstellen zum Schreiben auf lpr*/
   p = popen ("lpr", "w");
   if (p == NULL) {
      perror ("popen()");
      exit (EXIT_FAILURE);
   }
   /*An den Drucker schreiben */
   printf ("Sende Auftrag an den Drucker ...\n");
   fprintf (p, "%s", text);
   fflush (p);
   pclose (p);
   return;
}
```

```
int main (void) {
   char puffer[BUF], inhalt[BUF], antwort[BUF], pid[6];
   int r_fd, w_fd, n, i;
   inhalt[0] = '\0';
   /*Alle Zugriffsrechte der Dateikreierungsmaske
erlauben*/
   umask(0);
   if (mkfifo ("fifo1.1", O_RDWR | 0666) < 0) {
      /* FIFO bereits vorhanden - kein fataler Fehler */
      if(errno == EEXIST)
         printf ("Versuche, vorh. FIFO zu verwenden\n");
      else {
         perror("mkfifio()");
         exit (EXIT_FAILURE);
      }
   }
   /*Empfänger liest nur aus dem FIFO */
   r_fd = open ("fifo1.1", O_RDONLY);
   if (r_fd == -1) {
      perror ("open(1)");
      exit (EXIT_FAILURE);
   }
```

Beispielprogramme

```
/* Server weiter...*/
while (1) {      /*Endlosschleife */
      if (read (r_fd, puffer, BUF) != 0) {
         an_den_drucker (puffer);
         /*PID des aufrufenden Prozesses ermitteln */
         n = 0, i = 0;
         while (puffer[n] != '\n')
            pid[i++] = puffer[n++];
         pid[++i] = '\n';
         strcpy (antwort, "fifo.");
         strncat (antwort, pid, i);
         w_fd = open (antwort, O_WRONLY);
         if (w_fd == -1) {
            perror ("open(2)");
            exit (EXIT_FAILURE);
         }
         write (w_fd, TEXT, sizeof(TEXT));
         close (w_fd);
      }
      sleep(1);
   }
   return EXIT_SUCCESS;
}
```

Beispielprogramme

Ausführung:

```
$ gcc -o client client.c
$ gcc -o server server.c
$ ./server
[andere Konsole]
$ ./client
oder
$ ./server&
$ ./client
```