minitalk

METHODOLOGIE

Vous devez réaliser un programme de communication sous la forme d'un client et d'un serveur.

CONSIGNES

- Le serveur doit être lancé en premier et doit, après le lancement, afficher son PID.
- Le client prend deux paramètres :
  - Le PID du serveur.
  - Une chaîne de caractères à transmettre.

- Le client doit communiquer au serveur la chaîne passée en paramètre.
  Une fois la chaîne entièrement reçue, le serveur doit l'afficher.

- Le serveur doit être capable d'afficher la chaîne rapidement. Par rapidement, on entend que si vous pensez que c'est trop long, alors c'est sûrement trop long.

- Votre serveur doit pouvoir recevoir des chaînes de plusieurs clients à la suite sans nécessiter d'être relancé.

- La communication entre vos programmes doit se faire uniquement à l'aide de signaux UNIX.

- Vous ne pouvez utiliser que les deux signaux suivants : SIGUSR1 et SIGUSR2.

SEXTOS

Terminal A

```
→ minitalk ./server
server's PID : 14442
```

Terminal B

```
→ minitalk ./client 14442 "Hello, ça roule?"
message sent successfully to server 14422
```

Terminal A

```
→ minitalk ./server
server's PID : 14442
Hello, ça roule?
```

ALLOWED FUNCTIONS

minitalk

METHODOLOGIE

write()

`write()` prend en paramètres le `fd`, la chaîne de caractères, et la longueur de la chaîne.
- 0 = entrée standard
- 1 = sortie standard
- 2 = sortie d'erreur

ft_printf()

`ft_printf()` gère les `%x`, les `%s`, les `%d`, les `%i`, les `%u`, les `%p`, les `%c`, les `%%`
- Servira à imprimer les messages d'erreur
- Servira à imprimer l'accusé de réception du serveur

signal

`signal()` permet la communication système - processus ou processus - processus
- Définit un gestionnaire de signal
- Nombre de signaux définit par des macros (exemple SIGSEV = accès illégal à la mémoire)
- SIGUSR1 = signal user 1 / SIGUSR2 = signal user 2

sigemptyset

`sigemptyset()` Vide l'ensemble de signaux ajoutés par set

sigaddset

`sigaddset()` permet ajouts et suppressions

sigaction

`sigaction()` permet de dérouler le signal
- La structure de type struct sigaction contient les paramètres de déroulement
- Modifie l'action entreprise par un process à la réception du signal spécifique

kill

`Kill()` est destiné à envoyer tout types de signal à un ou plusieurs processus
- Le prototype de la fonction kill est int kill(pid_t pid, int signal);
- Si le pid est positif, le signal sig est envoyé au processus ou groupe de processus

getpid

`getpid()` sert à récupérer le pid d'un process (trèèès souvent mis dans printf)
- Très souvent présent dans la fonction printf avec un %d entre les guillemets

malloc

`malloc()` alloue dynamiquement de l'espace mémoire

free

`free()` désalloue/libère la mémoire dynamique allouée

pause

`pause()` met en pause le process jusqu'à la réception d'un signal

sleep

`sleep()` suspend l'exécution du programme durant un nombre donné de secondes

usleep

`usleep()` suspend l'exécution du programme durant un nombre donné de microsecondes

exit

`exit()` quitte le programme lorsqu'il reçoit le signal

INFOS ++

minitalk

METHODOLOGIE

PROCESSES

Pids

Process id

Utilisé pour connaitre l'id du père et de l'enfant

Pid > 0 = Père

Pid == 0 = Fils

Fork()

Divise le process en 2

On utilise pid pour connaitre les id du père et du fils

Getpid()

Affiche le pid du père ou de l'enfant

On le place dans la fonction printf après avoir écrit un %d

SIGNAUX

Signal()

Process to process (or system) communication

Gestionnaire de signal définit par des macros

SIGUSR1 = Client

SIGUSR2 = Server

Sigaction

Modifie l'action du process lors de la réception du signal

Struct Sigaction -> déroulement d'un signal

minitalk

METHODOLOGIE

LIENS UTILES

Fork, pid, signaux + SIGUSR1, SIGUSR2

https://www.youtube.com/watch?v=czhvXVZIXsM

Gestionnaire de signaux

https://www.youtube.com/watch?v=DWLoo1NQNz8&t=176s

Tout sur les PID, surtout waitpid

https://www.delftstack.com/fr/howto/c/waitpid-in-c/

Sigaction PDF

https://www.lri.fr/~mandel/systeme/systeme-06.pdf

Sigaction

http://manpagesfr.free.fr/man/man2/sigaction.2.html#:~:text=sa_handler%20indique%20l'action%20affect%C3%A9e,de%20signal%20comme%20seul%20argument.

Bitwise operators

https://www.youtube.com/watch?v=jlQmeyce65Q

Gestionnaire de signaux

https://www.youtube.com/watch?v=DWLoo1NQNz8&t=176s

Tout sur les PID, surtout waitpid

https://www.delftstack.com/fr/howto/c/waitpid-in-c/

Sigaction PDF

https://www.lri.fr/~mandel/systeme/systeme-06.pdf

Sigaction

http://manpagesfr.free.fr/man/man2/sigaction.2.html#:~:text=sa_handler%20indique%20l'action%20affect%C3%A9e,de%20signal%20comme%20seul%20argument.

PROJECT DEFENSE
CLIENT.C

minitalk

METHODOLOGIE

→ `int main(int ac, char **av)`

Initialisation du PID (process id)

Sécurité en cas de lancement du programme avec + ou - 3 arguments
`erreur - printf();`

Conversion du PID de ASCII à INT

Entrée dans la fonction `void sigmsg(char *msg, pid_t pid)`

La boucle while attend un signal pour lancer le programme

→ `void main(char *msg, pid_t pid)`

Initialisation d'un curseur (index i) à -1 parce que dans la boucle
on commence par ++i au lieu de créer la boucle while et mettre i++;
en fin de fonction

Tant qu'on entre le message en 3ème paramètre (`av[2]`)
Alors on entre dans la fonction `void ft_msg(char *c, pid_t pid)`
Dans laquelle on a envoyé caractère en dernière position `msg[i]` dans
la fonction précédente.

→ `void ft_msg(char c, pid_t pid)`

Initialisation d'un curseur (index i) à 8.

Tant que i se décrémente de 1 bit

partie sombre

if (c & (1 << i))
opérateurs « bitwise »
kill(pid, SIGUSR1);
envoie du signal
PID = process id
SIGUSR1 = permet d'envoyer un signal à l'user 1

else
kill(pid, SIGUSR2);
envoie du signal
PID = process id
SIGUSR1 = permet d'envoyer un signal à l'user 1
usleep(50);
Suspend l'exécution d'un process pendant
quelques milisecondes (50)

PROJECT DEFENSE
SERVER.C

minitalk

METHODOLOGIE

→ `int main(void)`

`struct sigaction siga;`
Initialisation d'une structure de signal avec le genre de traitement qu'il va recevoir à travers la fonction `sighandler`

→ `void main(char *msg, pid_t pid)`

Initialisation d'un curseur (index i) à -1 parce que dans la boucle on commence par ++i au lieu de créer la boucle while et mettre i++; en fin de fonction

Tant qu'on entre le message en 3ème paramètre (`av[2]`)
Alors on entre dans la fonction `void ft_msg(char *c, pid_t pid)`
Dans laquelle on a envoyé caractère en dernière position `msg[i]` dans la fonction précédente.

→ `void ft_msg(char c, pid_t pid)`

Initialisation d'un curseur (index i) à 8.

Tant que i se décrémente de 1 bit

partie sombre

if (c & (1 << i))
opérateurs « bitwise »
kill(pid, SIGUSR1);
envoie du signal
PID = process id
SIGUSR1 = permet d'envoyer un signal à l'user 1

else
kill(pid, SIGUSR2);
envoie du signal
PID = process id
SIGUSR1 = permet d'envoyer un signal à l'user 2

usleep(50);
Suspend l'exécution d'un process pendant quelques milisecondes (50)