On commence en douceur

Sous Unix, lancer la commande
man sscanf

Ecrire rnsuite une applcation en c permettant de saisir au clavier une chaine de caractères contenant un nom, un prénom et un numéro d'étudiant, par exemple
Rambo John 3922803
et doit en retour afficher sur la sortie standard
3922803 John Rambo
Un indice, **sscanf** est votre amir

Créer des processus avec fork

On rappelle que les processus sont tous gérés par le système d'exploitation. Les processus sont stockés dans la mémoire USER, mais c'est dans l'espace mémoire SUPERUSER que le système garde trace de tous les processus, dans une table nommée table des processus. La table des processus contient les contextes de tous les processus, c'est à dire tout ce qu'il faut conserver sur un processus pour le sauvegarder et le restaurer. Cette table est utilisée par l'ordonnanceur du système d'exploitation qui décide d'arrêter temporairement l'exécution d'un processus appelé processus sortant en stockant son contexte dans la case correspondante de la table des processus, de choisir un autre processus dans la liste des processus prêts disponibles (élection d'un processus), et resauration de ce processus entrant pour exécution.
La succession sauvegarde/élection/restauration est appelée changement de contexte et s'effectue au rythme du tick système.

Comment une application USER fait-elle pour ajouter un nouveau processus ?

Si on reformule cette question, cela revient pour une application USER de demander au système d'exploitation de rajouter une entrée supplémentaire dans la table des processus de l'espace SUPERUSER. Lorsqu'une application de l'espace USER demande un tel service au système d'exploitation, elle le fait au moyen d'un appel système. Une instruction privilégiée du processeur fait temporairement basculer du monde USER vers le monde SUPERUSER. Il est important de comprendre que l'application demande au système de faire quelque chose pour elle qui touche au système, et que le code machine qui réalise l'appel système se fait dans l'espace SUPERUSER sans l'application n'ait aucun contrôle. Un appel système rend une valeur de retour, qui indique à l'application si le service demandé a été correctement effectué. Une valeur de retour négative indique généralement l'échec de l'appel système.

L'appel système **fork**

Faire un
man fork
et lire la fonctionnement de l'appel système.

Quels sont les paramètres de **fork** ?

Que contient la valeur de retour de **fork** ?

Expliquer pourquoi l'appel système **fork** rend DEUX valeurs de retour.

En prenant l'exemple d'une application USER qui débute son tick système (donc il lui reste beaucoup de temps pour faire des choses) et qui fait un appel système **fork**, expliquer en détail ce qui se passe dans le système.

fork1

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

int main()
{

   pid_t p = fork();
   if(p<0){
     perror("fork fail");
     exit(1);
   }
   printf("Hello world!, process_id(pid) = %d \n",getpid());
   return 0;
}
```

Ecrire le même programme qui réalise un appel système **fork**. L'application doit clairement indiquer ce qui se passe dans la partie child et ce qui se passe dans le parent.

Fork2.c

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

int main()
{

   pid_t p = fork();
   if(p<0){
     perror("fork fail");
     exit(1);
   }
   else if (p==0)
   {
      printf("Hello world! from child, process_id(pid) = %d \n",getpid());
      exit(0) ;
   }
   else
```

```
    {
       sleep(4);
       printf("Hello world! from parent, process_id(pid) = %d \n",getpid());
    }
    return 0;
}
```

fork3.c

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

int main()
{

    int val;
    pid_t p = fork();
    if(p<0){
      perror("fork fail");
      exit(1);
    }
    else if (p==0)
    {
       printf("Hello world! from child, process_id(pid) = %d \n",getpid());
       exit(0);
    }
    else
    {
       scanf("%d",&val);
       printf("Hello world! from parent, process_id(pid) = %d \n",getpid());
    }
    return 0;
}
```

Un minishell

Que fait le code suivant ?

mini1.c

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

int     main()
{
```

```
	char	*buffer = NULL;
	size_t	buf_size = 2048;

	// alloc buffer qui stockera la commande entree par l'user
	buffer = (char *)calloc(sizeof(char), buf_size);
	if (buffer == NULL) {
		perror("Malloc failure");
		return (EXIT_FAILURE);
	}

	// ecriture d'un prompt
	write(1, "$> ", 3);

	// lecture de STDIN en boucle
	while (getline(&buffer, &buf_size, stdin) > 0) {
		printf("cmd = %s\n", buffer);
		write(1, "$> ", 3);
	}

	printf("Bye \n");
	free(buffer);
}
```

On donne maitenant un code beaucoup plus complet mini2.c avec LA fonction importante à écrire, **exec_cmd**.

mini2.c

```
void exec_cmd(char **cmd)
{
 }
```

Terminer votre minishell.

Les 4 appels système open, close, read, write

Le système d'exploitation gère donc les processus mais également d'autres ressources importantes, les fichiers des utilisateurs. Une application USER peut ouvrir (open) un fichier pour le lire (read) ou écrire dedans (write). Quand les opérations de lecture et d'écriture sont terminées, on ferme le fichier (close). Une des forces de Unix est que l'accès à un périphérique (imprimante par example) se fait au travers du fichier UNIX qui le réprésente /dev/parport0 par exemple.

Ainsi, imprimer revient à ouvrir (open) le fichier 'imprimante', à écrire dans ce fichier (write) pour réaliser l'impression, puis fermer le fichier (close)

Commandes à essayer

man 2 open

man 2 close

man 2 read

man 2 write

Pourquoi le nom **fd** ?

Que sont fd=0, 1, 2 ?

Ecritre votre propre version de la commande cp en utomosant les appels système ci-dessus ?

Le début du projet

Lire la règle du jeu.

Ecrire le code pour mélanger un paquet de 13 cartes.

Quelle est la stucture fondamentale à maintenir dans ce jeu ?

Ecrire le programme c permettant de construire cette structure en mémoire.

server_v1.c

Programmation graphique avec la bibliothèque SDL2

Le répertoire ex1 contient un exemple de programme SDL2 complet.

Compiler l’application.

Modifier le programme pour afficher plus de cartes.

Transformer le curseur marteau en cible.

Faire sorte de barrer tous les personnages qui ont un symbole crane.