Prog système

----------Threads----------

```c
struct thread_args {
   int *total;
   pthread_mutex_t *mutex;
};

void *thread_num_aleatoire(void *args) {
   struct thread_args *a = args;
   pthread_mutex_lock(a->mutex);
   *(a->total) += 10;
   pthread_mutex_unlock(a->mutex);
}

int main() {
   int total = 0;
   pthread_mutex_t mutex;
   pthread_mutex_init(&mutex, NULL);
   struct thread_args args = {
       .total = &total,
       .mutex = &mutex,
   };
   pthread_t t1, t2;
   pthread_create(&t1, NULL,
thread_num_aleatoire, &args);
   pthread_create(&t2, NULL,
thread_num_aleatoire, &args);
   printf("Total : %d\n", *(args.total));
}
```

----------Pipe----------

```c
int main() {
   int fds[2];
   pipe(fds);
   int entree = fds[0];
   int sortie = fds[1];
   if (fork() == 0) {
       close(entree);
       write(sortie, "Hello world", 12);
       close(sortie);
       exit(EXIT_SUCCESS);
   }
   close(sortie);
   char tmp[20];
   read(entree, tmp, 12);
   printf("Reçu : %s\n", tmp);
   exit(EXIT_SUCCESS);
}
```

----------Redirections----------

```c
int main() {
   int fdInput = open("input.txt", O_RDONLY);
   dup2(fdInput, STDIN_FILENO);
   close(fdInput);
   int fdOut = open(
       "output.txt",       // path
       O_CREAT | O_WRONLY, // flags
       0644                // perms
   );
   dup2(fdOut, STDOUT_FILENO);
   close(fdOut);
   execlp("tr", "tr", "[a-z]", "[A-Z]", NULL);
}
```

----------Signaux----------

```c
void sig_handler(int sig) { // Appelée quand le
processus est intérrompu
   printf("recue SIGINT\n");
   exit(0);
}

int main(void) {
   signal(SIGINT, sig_handler); // Enregistre
la fonction
   while (1) sleep(1);
}
```

----------Signaux----------

```c
void sig_handler(int sig) {
   printf("Reçu SIGUSR1 de la part de %d\n",
sig);
   exit(0); // on stop le programme dès que
l'on a terminé
}

int main(void) {
   // Inscription signal handler
   signal(SIGUSR1, sig_handler);
   pid_t parent_pid = getpid();
   // L'enfant emet un signal vers le père
   if (fork() == 0)
       kill(parent_pid, SIGUSR1);
   while(1) sleep(1); // Attente infinie
}
```

----------Exec----------

```c
int main() {
   if(fork()==0) // Dans un fils car appel bloquant
       execl("/bin/ls", "ls", "-l", NULL);
   // peut faire ses bails sans être bloqué grace au fork
}
```

----------Fork & Waitpid----------

```c
int main() {
   pid_t p = fork();
   if (p == 0)
       exit(EXIT_FAILURE);
   int status;
   waitpid(p, &status, 0);
   printf("le s'est terminé avec le status
%d\n", status);
}
```

----------exec----------

fonction	Particularité	exemple
`execl`	path + liste args	`execl ("/bin/ls", "ls", "-l", NULL)`
`execlp`	cmd + liste args	`execlp( "ls", "ls", "-l", NULL)`
`execv`	path + array[] args	`execv ("/bin/ls", arg)`
`execvp`	cmd + array[] args	`execvp( "ls", arg)`