Compte Rendu TP1

ELHIMDI Yasmine

PARMENTIER Laurent

Exercice 1.1

Source

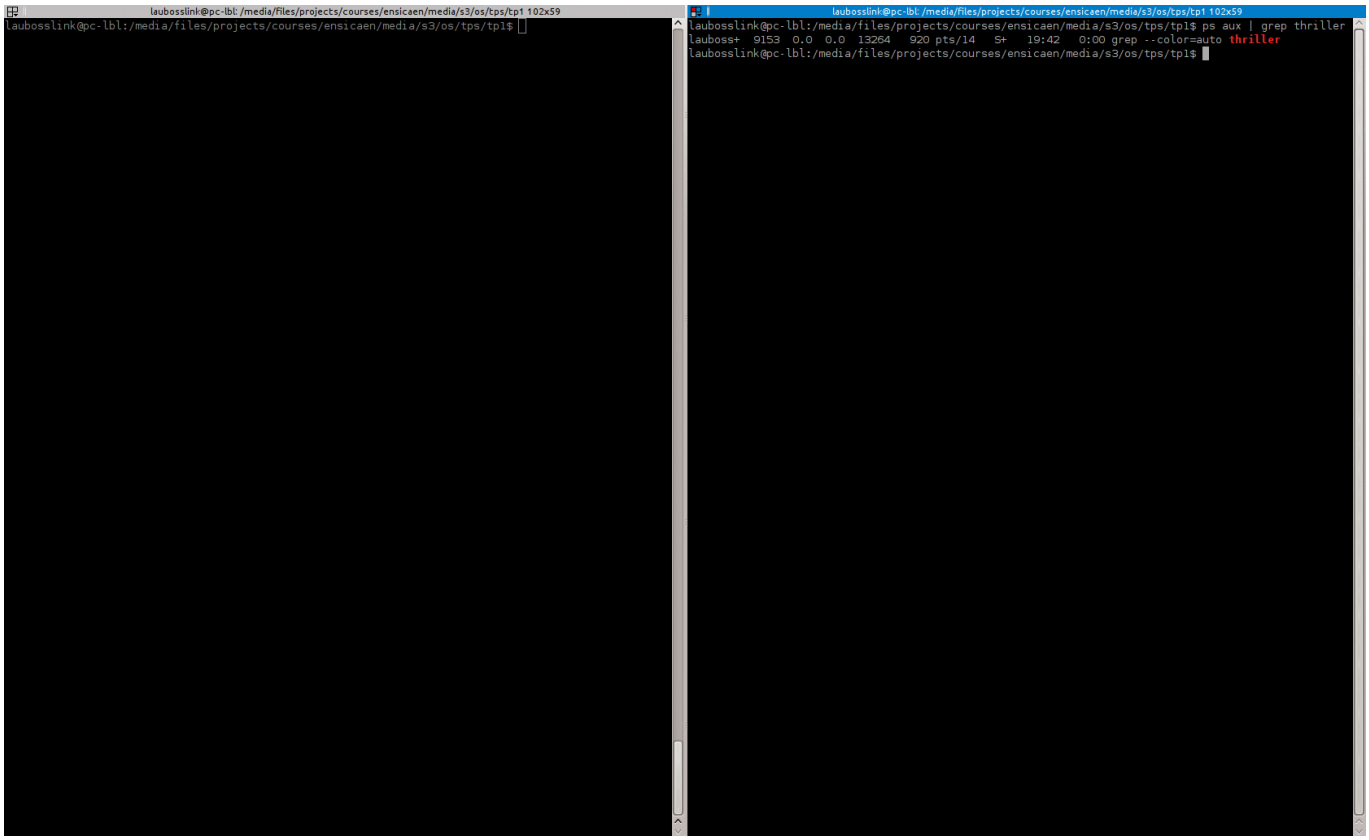
thriller.c

- thriller.c

```
#include <stdio.h> /* printf() */
#include <unistd.h> /* fork(), perror() */
#include <stdlib.h> /* exit() */

int main()
{
    pid_t pid, wait_pid;
    int status;

    pid = fork();


    if(pid == -1)
        perror("fork() can't be created");

    /* I'm child */
    if(pid == 0)
    {
        printf("%d: I'm child !\n", getpid());
        exit(2);
    } else
    {
        printf("%d: I'm father, and i create child with pid '%d'\n",
getpid(), pid);
        pause();

        do {
```

```
            /* retrieve status of any child  */
            wait_pid = wait(&status);

            if(wait_pid == -1)
            {
                perror("wait() error");
                exit(1);
            }

            /* Check child is ending */
            if(WIFEXITED(status))
                printf("%d: retrieve pid(%d), killed with status %d\n",
getpid(), wait_pid, WEXITSTATUS(status));

        } while(!WIFEXITED(status));

        printf("%d: End of father processus.\n", getpid());
        exit(EXIT_SUCCESS);
    }
}
```

CLI



thriller.gif

Exercice 1.2

Source

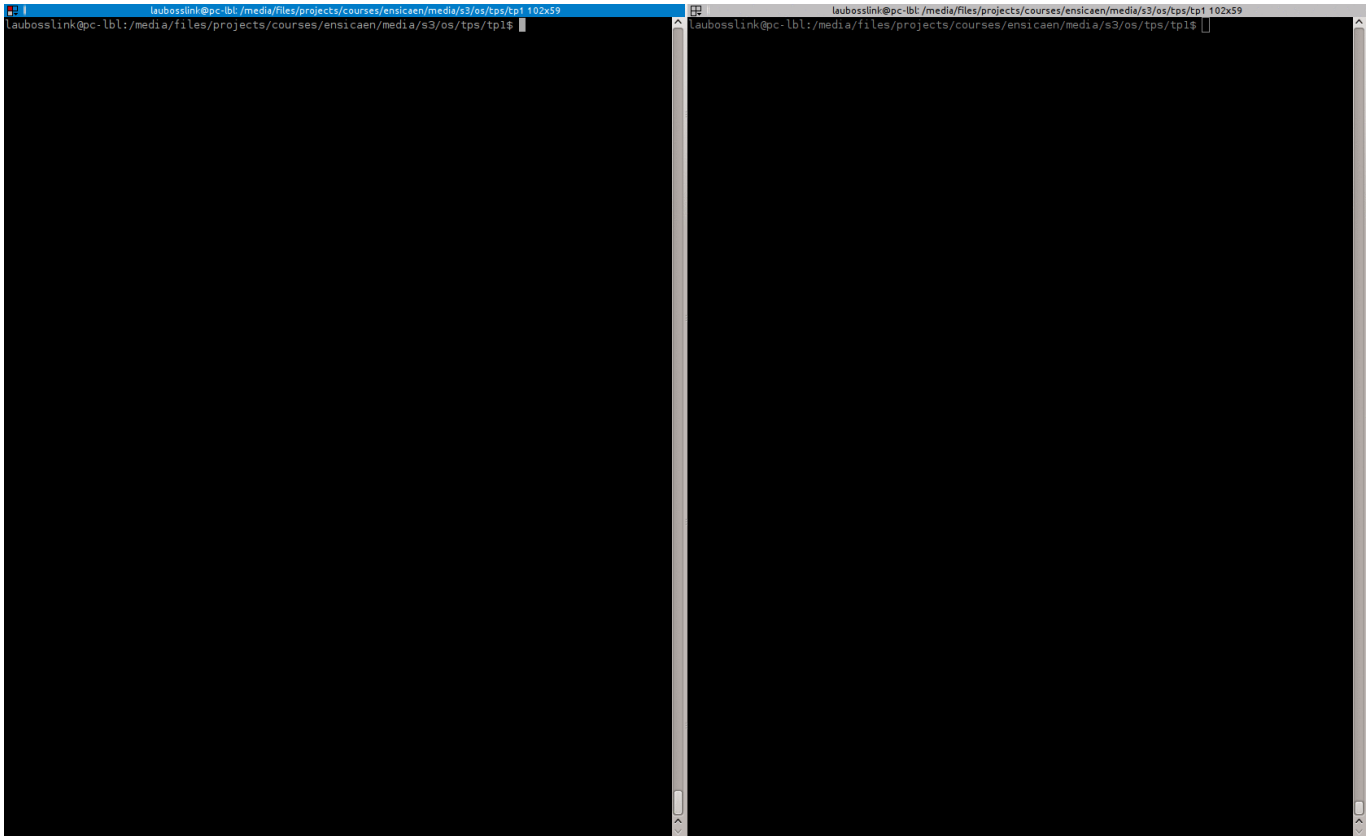
fork_date.c

- fork_date.c

```
#include <stdio.h> /* printf() */
#include <unistd.h> /* fork(), perror() */
#include <stdlib.h> /* exit() */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
    pid_t pid, pid2, wait_pid;
    int i, status;

  // message.log need to exist (rm message.log; touch message.log)
  int fd = open("message.log", O_RDWR);

    pid = fork();

    if(pid == -1)
        perror("fork() can't be created");

    /* I'm child */
    if(pid == 0)
    {
    close(1); // close stdout
    dup(fd); // redirect stdout to fd

        for(i=0; i<10; i++)
        {

            pid2 = fork();

            if(pid2 == 0)
            {
                int res = execl("/bin/date", "date", "-u", (char*)
NULL);

        if(res < 0)
```

```
                {
                    printf("error execl\n");
                    exit(res);
                }

        exit(0);
            }

      sleep(3);
        }

    close(fd);
    exit(0); /* really important, else child continue */
    }

    /* father create second child */
    pid2 = fork();

    if(pid2 == -1)
        perror("fork() 2 can't be created");

    /* i'm second child */
    if(pid2 == 0)
    {
    close(1);
    dup(fd);

        for(i=0; i<30; i++)
        {
            printf("Attendre !\n");
      fflush(stdout);
            sleep(2);
        }

    close(fd);
    exit(0);
    }

  /* '(int*) NULL' is important, could also specify 'int status' var.
*/
  while(wait((int*) NULL) != -1); /* wait all childs */

  printf("C'est terminé!.\n");
}
```

CLI



fork.gif

Question

Explicitez la difference de placer la fonction dup dans les deux processus fils au lieu du processus pere.

Réponse: Si on plaçait la fonction dup dans le processus père, on redirigerai la sortie standard du père dans le file descriptor. Or on veut rediriger la sortie standard des deux fils vers un fichier.

Ce résultat est bien vérifié par la présence de “C'est terminé” dans le terminal, et non dans le fichier message.log.