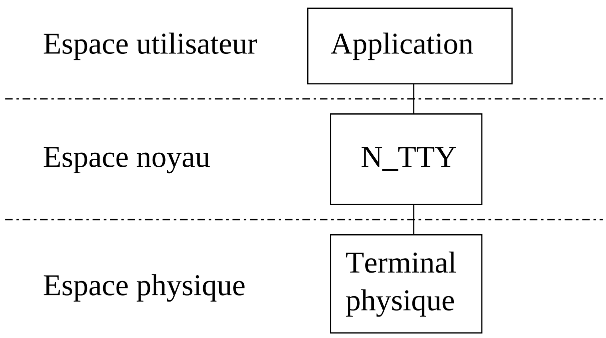
Terminaux et Pseudo-terminaux

1 Terminaux

Des vrais terminaux, il n'y en a finalement que peu dans un système Linux : en général, il n'y a que les consoles textes qui sont de « vrais » terminaux, i.e. qui sont gérés complètement par le noyau : lorsque l'on appuie sur Control-C, le noyau a tous pouvoirs pour envoyer un SIGINT aux processus en avant-plan.

Qu'est-ce qu'un terminal ? C'est long à décrire. En un mot, c'est ce sur quoi un processus « vit » : on dit que c'est son **terminal de contrôle**. On le voit dans la colone TTY de la commande `ps`. Même si entrée standard, sortie standard et sortie d'erreur sont redirigées vers autre chose, tout fils héritera du terminal de contrôle. **Vérifiez-le** avec un `sleep 1h &`

Les mécanismes fournis par un terminal sont nombreux. Pour un aperçu, lancer la commande `stty -a` : apparaît alors la configuration du terminal. Celle-ci est remplie de choses diverses, héritées des périphériques matériels qu'on utilisait naguère comme terminaux : vitesse et parité du port série, taille de l'écran, caractères utilisés pour le contrôle des processus (Control-C et compagnie), politique de flush de la ligne, contrôle des lignes du modem, le contrôle de flux, configuration des tabulations, contrôle de l'édition en ligne (ce qui permet de taper backspace après avoir lancé `cat` sans que ce dernier ait à s'en soucier)... Tout cela est géré par le noyau dans la couche `N_TTY` :



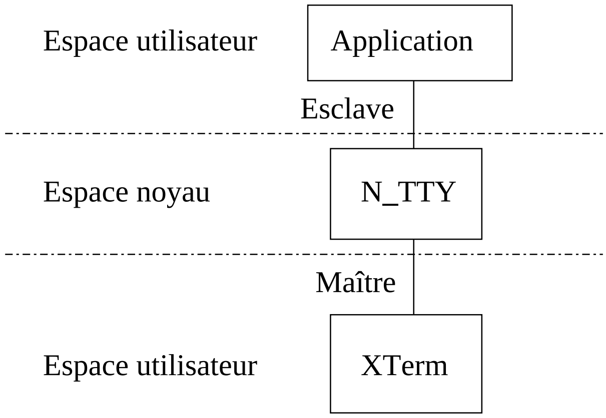
Ce qui nous intéressera dans ce TP, ce sont les les caractères utilisés pour le contrôle. Dans le cas d'un vrai terminal, quand on appuie sur Control-C, celui-ci envoit un caractère 3 (Control-C) au noyau, qui envoie alors un SIGINT à l'application, quelle qu'elle soit :

cf la fonction `n_tty_receive_char()` dans le noyau pour les détails de traitement des caractères.

Mais dans le cas d'un XTerm (ou gnome-terminal, konsole, ...), quand on appuie sur Control-C, c'est XTerm, i.e. un programme utilisateur, qui récupère l'évènement, or « dans » le XTerm on peut lancer `su` pour lancer des applications sous une autre identité, et Control-C permet pourtant bien d'interrompre ces programmes lancés sous d'autres identités (et donc XTerm ne peut a priori pas les tuer) !

2 Pseudo-Terminaux

XTerm, screen, ssh, ... Ces utilisent les pseudo-terminaux. Le principe est tout simple : on remplace le terminal physique par un terminal virtuel :



Ce qu'on appelle « pseudo-terminal », c'est l'ensemble { côté Maître, `N_TTY`, côté Esclave }. XTerm récupère les pressions touches depuis le serveur X, les envoie du côté Maître, et c'est le noyau qui agit alors en conséquence du côté esclave (tuer l'application par exemple).

On va réaliser un programme qui lance une application sur le côté esclave d'un pseudo-terminal.

Vous pourrez lire `man pts` et autres pages de manuel utiles...

Dans un premier temps, écrivez un programme qui crée un pseudo-terminal :

- Ouvrir le fichier `/dev/ptmx` en lecture/écriture et avec les flags `O_NOCTTY` | `O_RDWR`.
- Appeler `grantpt` puis `unlockpt` dessus.
- Appeler `ptsname` dessus pour récupérer le chemin de la partie esclave du pseudo-tty.

Appelez alors `fork` pour créer un processus fils.

- dans le fils, fermer `STDIN_FILENO`, `STDOUT_FILENO` et `STDERR_FILENO`, appeler `setsid()` pour rendre le fils indépendant de son père et le détacher du terminal du père, ouvrir le côté esclave (qui devient donc son entrée standard), et enfin appeler `system("cat > toto");` : l'application ici sera donc un simple programme cat.

– dans le père, écrire sur le côté maître `"toto\n"`, faire un `sleep()` d'une dizaine de secondes, écrire un caractère 3 (Control-C) et appeller finalement `pause()`.

Observez dans ps que `cat` meurt effectivement au moment où le caractère 3 est envoyé (*i.e.* au bout d'une dizaine de secondes). Au passage, Si l'on omet `\n` dans le write du côté maître, que se passe-t-il ? Pourquoi ?

3 Taille de terminal

Exécutez `strace -o ˜/log nano` dans un XTerm, et `tail -f ˜/log` en parallèle dans un autre XTerm. Redimensionnez le premier XTerm. Nano reçoit des signaux `SIGWINCH` ! Il utilise ensuite `ioctl(fd, TIOCGWINSZ, &win);` pour récupérer la nouvelle taille.

Dans le programme de la section précédente, ajoutez, du côté maître, un appel à `ioctl(fd, TIOCSWINSZ, &win)` (`win` est une `struct winsize` dont les champs sont `ws_row`, `ws_col`, `ws_xpixel` et `ws_ypixel`). Côté client, ajoutez `strace -f ~/log` dans l'appel à `cat`. Constatez que `cat` reçoit effectivement un `SIGWINCH`.

4 Encore plus fort

Réfléchissez au cas de `ssh` : où le pseudo-terminal est-il créé ? Comment les commandes de contrôles sont-elles propagées et interprétées ?

Mêmes questions pour `screen`.

Après, on peut imaginer un screen dans un ssh dans un screen dans un xterm, ...