Minishell

• Commandes autorisées dans le projet

**write** -> écrire dans un descripteur de fichier

**malloc** -> alloue de la mémoire

**free** -> libérer la mémoire allouée

**printf** -> afficher une valeur (%s, %d …)

**access** -> vérifier les permissions utilisateurs d’un fichier

**open** -> ouvrir un file descriptor

**read** -> lire un file descriptor

**close** -> fermer un file descriptor

**fork** -> creer un processus fils

**wait** -> attendre qu’un processus change d'état

**waitpid** -> attendre qu’un processus change d'état (avec son pid)

**wait3**

**wait4**

**signal** -> gestion de signaux

**sigaction** -> examiner et modifier l’action associé à un signal

**sigemptyset** -> initialise un ensemble de signaux à l'ensemble vide. Tous les signaux reconnus sont exclus.

**sigaddset** -> ajouter un ensemble de signaux

**kill ->** envoyer un signal à un processus

**exit** -> sortir d’un programme

**stat** -> obtenir l'état d’un fichier

**lstat** -> donne l'état du lien au lieu du fichier visé si c’est un lien symbolique

**fstat** -> identique à stat, sauf que le fichier est ouvert par un fd

**unlink** -> détruit un nom et éventuellement le fichier associé

**execve** -> exécuter un programme

**dup** -> dupliquer un file descriptor

**dup2** -> dupliquer un file descriptor (duplique le old fd dans le new fd)

**pipe** -> créer un tube pour que deux processus puissent communiquer

**getcwd** -> obtenir le répertoire de travail courant

**chdir** -> changer de répertoire courant (man 2 chdir)

**opendir** -> ouvrir un répertoire

**readdir** -> consulter un répertoire

**closedir** -> fermer un répertoire

**readline** -> afficher un prompt

**rl\_clear\_history** -> supprimer l’historique

**rl\_on\_new\_line** -> indiquer que nous sommes passés à une nouvelle ligne (vide), généralement après un \n.

**rl\_replace\_line ->**

**rl\_redisplay** -> change ce qui est affiché à l'écran pour refléter le contenu actuel de rl\_line\_buffer

**add\_history** -> ajouter un historique

**strerror** -> renvoie une chaîne de caractère avec le code d’erreur passé en argument

**perror** -> affiche le message d’erreur sur la sortie d’erreur

**isatty** -> vérifier si un fd se rapporte à un terminal

**ttyname** -> obtenir le nom d’un terminal

**ttyslot** -> trouver l'emplacement du terminal en cours dans un fichier

**ioctl** -> contrôler les périphériques

**getenv** -> récupérer le contenu d’une variable d’environnement

**tcsetattr**, **tcgetattr** -> configuration de terminal

**tgetent** -> charge l'entrée pour nom

**tgetflag** -> récupère l'entrée booléenne pour id

**tgetnum** -> récupère l'entrée numérique de id

**tgetstr** -> renvoie l'entrée de la chaîne de caractères pour id

**tgoto** -> instancie les paramètres dans la capacité donnée. La sortie de cette fonction doit être passée à tputs

**tputs**-> afficher la chaîne retournée.

Partie obligatoire

Ce qu’on gère :

-> builtins(env, pwd, export, unset, cd, exit, echo [-n])

-> commandes en execve (ls, cat, wc, head, tail, uname …)

Ce qu’on gère pas

-> messages d’erreur pour quotes non fermés, point virgule (sauf entre ““) et backslash

—-----------—-----------—-----------—-----------—-----------—-----------—-----------—-----------—-----------—-----------—-----------—-----------—--------------------

MAIN

• INIT

• LOOP

• (Entree commande)

• PARSING

• EXECUTION (cmd… ou exit)

• GESTION SIGNAUX CTRL-D (quitte) / CTRL-C (nouveau prompt)

• FREE

• return (0);

—----------------------------------------------------------------------------------------

INIT:

• structure variables ‘globales” (shell->)

• liste chainee “input”

• liste chainée 'env': shell->env ->name

->value

->exists

->exported

PARSING

• parser “input” => creer liste chainee => ptr sur liste

EXECUTION

• parcourir liste chainee “cmdlist”

• 1/ si = exit => break de la boucle main

• 2/ etc