Compte-rendu TP3 : DEBARD Jean et POIRIER Vincent

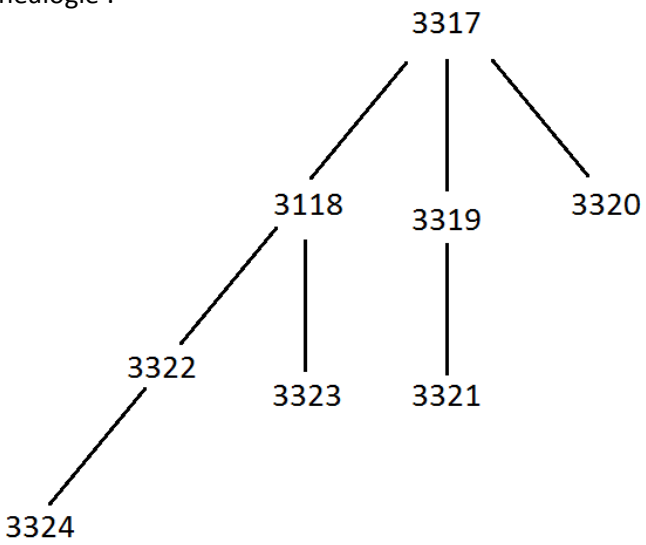
Exercice 1 : un peu de généalogie

- Sortie obtenue :

  Mon PID est 3317 et i=0
  Mon PID est 3317 et i=1
  Mon PID est 3317 et i=2
  Mon PID est 3320 et i=2
  Le processus 3320 a fini de s'executer
  Mon PID est 3319 et i=1
  Mon PID est 3318 et i=0
  Mon PID est 3319 et i=2
  Mon PID est 3318 et i=1
  Mon PID est 3318 et i=2
  Mon PID est 3322 et i=1
  Mon PID est 3322 et i=2
  Mon PID est 3321 et i=2
  Le processus 3321 a fini de s'executer
  Le processus 3319 a fini de s'executer
  Mon PID est 3323 et i=2
  Le processus 3323 a fini de s'executer
  Mon PID est 3324 et i=2
  Le processus 3324 a fini de s'executer
  Le processus 3322 a fini de s'executer
  Le processus 3318 a fini de s'executer

- 8 processus : 3317 – 3320 – 3319 – 3318 – 3322 – 3321 – 3323 – 3324

- Généalogie :



Exercice 2 : écriture d'un shell basique

```
#include "ligne_commande.h"

#include <stdlib.h>
#include <string.h>
#include <unistd.h> // fork(2),getpid(2)
#include <stdio.h> // perror(3),printf(3)
#include <sys/types.h> // getpid(2),wait(2)
#include <sys/wait.h> // wait(2)

int main(int argc, char** argv) {

	pid_t pid;
	int etat;

	while(1) {
		printf("> ");
		fflush(stdout);
		char** commande = lis_ligne();

		if(fin_de_fichier(commande)) {
			printf("Fin du programme...\n");
			return 0;
		} else {
			if(!ligne_vide(commande)) {
				if(!strcmp(commande[0], "exit")) {
					printf("Fin du programme...\n");
					return 0;
				}

				pid = fork();
				if(pid == 0) {
					if(execvp(commande[0], commande) == -1) {
						printf("Erreur, commande
incorrecte.\n");
					}
					exit(0);
				} else {
					wait(&etat);
				}
			}
		}
	}
}
```

Exercice 3 : gestion de l'environnement dans le shell

```
#include "ligne_commande.h"

#include <stdlib.h>
#include <string.h>
#include <unistd.h> // fork(2),getpid(2)
#include <stdio.h> // perror(3),printf(3)
#include <sys/types.h> // getpid(2),wait(2)
#include <sys/wait.h> // wait(2)

int commandes_internes(char** cmd);

int main(int argc, char** argv) {

	setenv("INVITE", "> ", 1);
	pid_t pid;
	int state;

	while (1) {
		printf("%s", getenv("INVITE"));
		fflush(stdout);
		char** commande = lis_ligne();

		if (fin_de_fichier(commande)) {
			printf("Fin du programme...\n");
			return 0;
		} else {
			if (!ligne_vide(commande)) {
				if (!commandes_internes(commande)) {
					pid = fork();
					if (pid == 0) {
						if (execvp(commande[0], commande) == -
1) {
							perror("Erreur, commande
incorrecte.\n");
							exit(0);
						}
					} else {
						wait(&state);
					}
				}
			}
		}
	}
}
```

```
int commandes_internes(char** cmd) {

	if (!strcmp(cmd[0], "exit")) {
		printf("Fin du programme...\n");
		exit(0);
	}

	if (!strcmp(cmd[0], "cd")) {
		if (cmd[1]) {
			if (chdir(cmd[1]) == -1) {
				perror("Error");
			}
		} else {
				perror("Vous n'avez pas saisi d'arguments.\n");
		}

		return 1;
	}

	if (!strcmp(cmd[0], "export")) {
		if (cmd[1]) {
			char* value = separe_egal(cmd[1]);
			if (setenv(cmd[1], value, 1) == -1) {
				perror("Error");
			}
		}

		return 1;
	}

	return 0;
}
```

- La commande export doit être implémentée en tant que commande interne (à l'intérieur de notre shell), car sinon les variables d'environnement écrites grâce à cette commande seraient définitivement « inscrites » dans le registre des variables d'environnement.

- Travailler avec des commandes internes permet de s'assurer que ce qu'on écrit sera effacé une fois le shell quitté.