TP2

Q1)

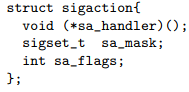
Le programme affiche “div 0” en boucle. On rajoute “ longjmp(env,0);” au programme pour qu’il s’exécute correctement.

setjmp/longjmp permet la gestion d’erreur ( ~ try/catch)

setjmp(env) sauvegarde le contexte de la pile et l'environnement dans la variable env

longjmp(env,0) récupère le contexte de la pile et l’environnement sauvegardé dans env. Une fois longjmp exécuté, le programme se poursuit comme si l’appel correspondant à setjmp venait de renvoyé le paramètre passé dans la fonction longjmp ( 0 ici )

Attention, le processus appelant setjmp ne doit pas se terminer avant longjmp.



int sigaction(int sig, const struct sigaction \*\_\_restrict\_\_ new, struct sigaction \*\_\_restrict\_\_ old);

int sig -> id du signal

\*new -> nouvel action a associé à sig ( si NULL reste le même action )

\*old -> stock l’action de sig avant application de la fonction

on créer la sigaction action et au sa\_handler on associe la fonction handler qui est le traitement lorsqu’une exception est levée

sigaction(SIGFPE,&action, NULL);

Récupérer le signal SIGFPE lui associe l’action def par handler et ne sauvegarde pas l’action précédente.

Q2)

sigset\_t -> ensemble de signaux ( masques )

int sigemptyset(sigset\_t \*set) -> initialise un set vide

int sigaddset(sigset\_t \*set, int signal) -> ajoute le signal au set

int sigprocmask(int option, const sigset\_t \*\_\_restrict\_\_ new\_set, sigset\_t \*\_\_restrict\_\_ old\_set)

Sig\_block -> block les signaux

new\_set -> signaux sur lesquels appliquer le chgmt

old\_set -> enregistrer le signal set dans son état initiale

int sigsuspend(const sigset\_t \*mask) -> Remplace le masque actuel du thread par le masque passé en paramètre puis suspend l'exécution du thread appelant

pendant sigsuspend tout passe (ancien masque qui laisse tout passer)

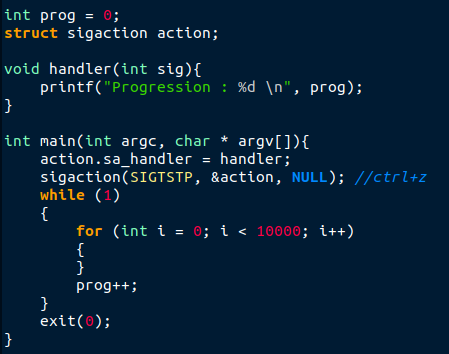
Q3)



Q4)

| attente active | Avec les masques |
| --- | --- |
|  |  |

Q5)



Q6)



lors d’un ctrl+z : , on voit bien qu’un grand nombre de signaux sont perdu