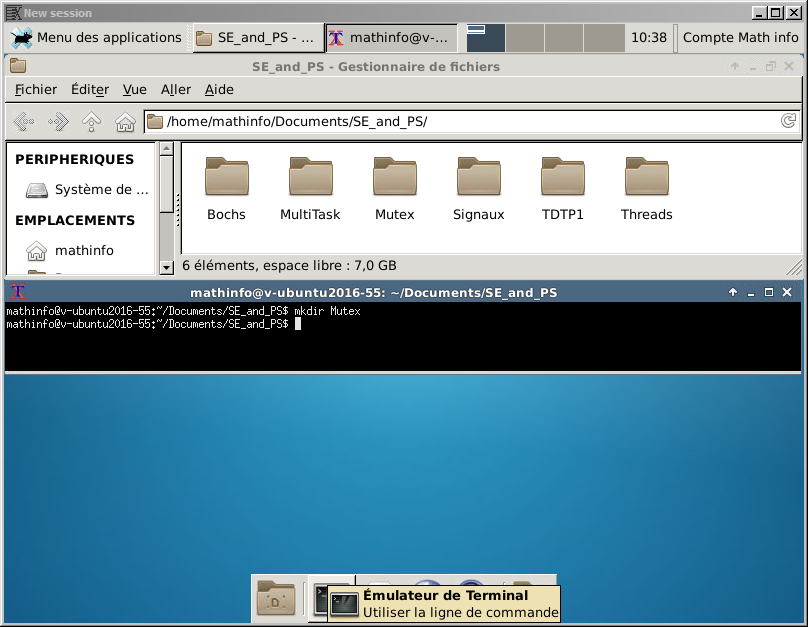
# 1516 L2 SEPS TDTP 6 – Threads Unix - Mutex

## I] Connexion



Nous pouvons voir sur l’image précédente, la connexion établie de la session.

## II] Le fond de l’air est frais

1] Dans le fichier "laChenille.c", écrivez un programme en C qui déclare une liste chainée à partir d’une structure Job qui contient un pointeur vers un Job, un entier jobIndex indiquant le numéro de ‘Job’, et deux entiers a et b, le ‘Job’ a réaliser étant d’effectuer la somme des deux nombres (qui seront tires au hasard) et de l’afficher a l’écran.

typedef struct Job{

int a;

int b;

int jobIndex;

struct Job \*svt;

}Job;

typedef struct{

Job\* pfirst;

// autre élément

}List;

2] Créez une fonction createJobs qui renvoie un pointeur sur un nouveau Job tire au hasard (mais avec un numéro de Job qui suit ceux des jobs précédents).

Job\* createJob(){

Job \*job = (Job\*) malloc(sizeof(Job));

job->a = RANDOM(5,2000);

job->b = RANDOM(5,2000);

job->jobIndex = 0;

job->svt = NULL;

return job;

}

3] Créez une fonction createJobs qui génère une liste chainée sur un nombre n de jobs passe en paramètre a la fonction et renvoie un pointeur sur le premier élément de la liste.

Job\* createListJobs(int n){

int i;

Job \*pfirst, \*cur;

pfirst = cur = createJob();

if(pfirst != NULL){

cur->jobIndex = 1;

for(i=1;i<n;++i){

cur->svt = createJob();

cur->svt->jobIndex = cur->jobIndex+1;

cur = cur->svt;

}

}

return pfirst;

}

4] Créez une fonction de thread providerFunction qui fait passer un entier variable globale flag a 1 une fois qu’il a entamé son exécution, puis qui attend se manière sécurisée d’être notifiée de l’absence de jobs a traiter, et qui, quand notifiée rajoute un nombre de jobs a la liste compris entre NB\_JOBS\_REFILL et 2xNB\_JOBS\_REFILL.

void\* providerFunction (void\* param){

List \*list = (List\*) param;

flag = 1;

while(1){

printf("Provider trying to lock Mutex ... ");

pthread\_mutex\_lock(&(list->mutexJob));

printf("Mutex locked by provider.\n");

pthread\_cond\_wait(&list->condNbrJob,&list->mutexJob);

printf("Asking for refill ... Provider activation !\n");

if (list->pfirst == NULL){

int refill = RANDOM(NB\_JOB\_REFILL,2\*NB\_JOB\_REFILL);

list->pfirst = createListJobs(refill);

printf("Provider : Added %d jobs to the job list !\n",refill);

}

pthread\_cond\_signal(&list->condWorker);

printf("Provider : Notifying workers.\n");

pthread\_mutex\_unlock(&list->mutexJob);

printf("Provider : Releasing Mutex.\n");

}

}

5] Créez une fonction de thread workerFunction qui prend de manière sécurisée le prochain job de la liste ; si aucun job n’est disponible, elle notifie le provider et attend d’être notifiee de la presence de jobs dans la liste. Apres avoir effectue et etre sortie de sa section critique, elle effectue le travail du job et attend pendant un temps compris entre 1 secondes et 1+SLEEP\_TIME secondes avant de recommencer.

void\* workerFunction (void\* param){

Parameters \*parameters = (Parameters\*) param ;

while(1){

printf("Worker trying to lock Mutex ... ");

pthread\_mutex\_lock(&(parameters->list->mutexJob));

printf("Mutex locked by worker.\n");

if (parameters->list->pfirst == NULL){

pthread\_cond\_signal(&parameters->list->condNbrJob);

pthread\_cond\_wait(&parameters->list->condWorker,

&parameters->list->mutexJob);

}

printf("Extracting job...");

Job \* extract = extractJob(parameters->list);

printf("Job obtained, releasing Mutex\n");

pthread\_mutex\_unlock(&parameters->list->mutexJob);

printf("-----------------------------------------\n");

printf("Mutex released, processing job:\n");

makeMyJob(extract);

printf("-----------------------------------------\n"); sleep(RANDOM(1,1+SLEEPTIME));

}

}

6] Creez un programme principal (main) qui initialise les differente structures et variables necessaires (toutes ne sont pas decrites dans l’enonce) a l’execution du programme, puis cree un thread a partir de la fonction providerFunction ; puis le main attend que le flag, initialement a zéro, passe a 1 pour commencer a generer un nombre NB\_WORKERS de threads bases sur la fonction workerFunction ; enfin, pour la forme, le programme principal attend la fin de l’execution des threads worker puis du provider.

int main(int argc, char \*argv[]) {

int ret,i;

List list;

list.pfirst = NULL;

Parameters parameters[NB\_WORKERS];

pthread\_mutex\_init(&list.mutexJob, NULL);

pthread\_cond\_init(&list.condNbrJob, NULL);

pthread\_cond\_init(&list.condWorker, NULL);

printf("Creation of provider's thread: \n");

ret = pthread\_create(&list.provider,NULL,&providerFunction, &list);

if(!ret){

printf("ok \n");

while(!flag);

printf("creation of workers thread: \n");

for(i=0;i<NB\_WORKERS;++i){

parameters[i].threadIndex = i+1;

parameters[i].list = &list;

printf("Creating worker's thread n%c%d... ",'°',i+1);

ret = pthread\_create(&list.worker[i],NULL,&workerFunction, &parameters[i]);

if(ret) printf("Error !\n");

else printf("OK !\n");

}

}

else

printf("Error creating provider's thread.\n");

for(i=0;i<NB\_WORKERS;++i){

pthread\_join(list.worker[i],NULL);

}

pthread\_join(list.provider,NULL);

return EXIT\_SUCCESS;

}

## Démonstration

## F:\UpfInfo\1516_UPF_L2_SEPS\1516_L2_SEPS_TDTP_6_STEGEN_THOMAS\pictures\2.PNG