

SYSTÈMES TEMPS RÉEL ET EMBARQUÉ

---

# Le Temps Réel Sous LINUX L'API POSIX

---

Réalisé par :

Hanane CHRIF EL ASRI  
Achraf BOUCHOUIK  
Khalid EL GAMOUS

Professeur :

Abdeslam EN-NOUAARY

## 1 *Introduction*

Dans ce travail, nous allons apprendre le développement en temps réel avec l'interface de programmation standard POSIX qui définit un ensemble de normes. La norme présente principalement une bibliothèque riche pthreads qui facilite la programmation concurrentielle ainsi que les aspects s'y relatant tels que la communication et la synchronisation. Nous allons voir plusieurs exemples pour bien comprendre les différents mécanismes utilisés.

POSIX est une famille de normes techniques définie depuis 1988 par Institute of Electrical and Electronics Engineers (IEEE).

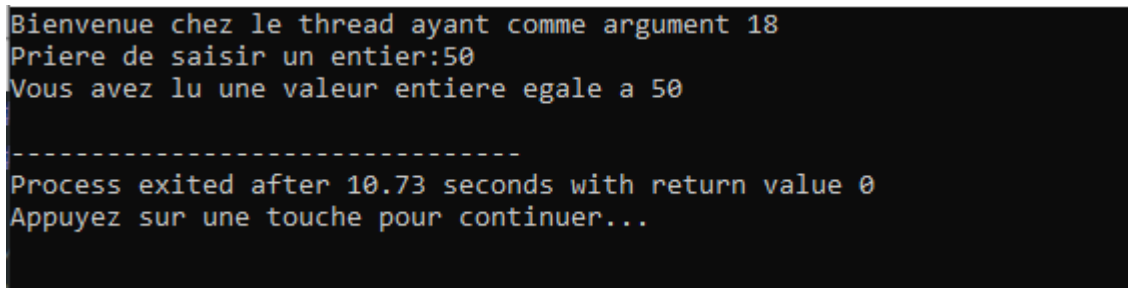
Dans ce TP, nous allons voir la bibliothèque pthreads de la norme POSIX qui nous fournit un ensemble de fonctions intéressantes pour la programmation temps réel surtout la concurrence/multitâches, et la communication et la synchronisation entre les tâches.

## 2 *Les threads POSIX*

Commençons par une illustration de la gestion des threads POSIX, nous allons construire une fonction qui reçoive le numéro de thread créé, ce dernier permet d'afficher un message de bienvenue et de lire un entier au clavier, puis de le retourner au thread principal qui est main.

Le thread principal initialise une nouvelle variable pour créer notre deuxième thread avec la fonction **pthread\_create** qui prend comme paramètres l'adresse du thread, la fonction **lire\_entier** qui va lire un entier et l'argument passé à cette fonction lors de l'appel.

Si la création est effectuée correctement, nous allons attendre la fin de l'exécution de notre thread par la fonction **pthread\_join** qui permet de stocker la valeur retournée par le thread dans une variable qu'on a créé dans le thread principal. Enfin, nous allons afficher la valeur lue.



```
Bienvenue chez le thread ayant comme argument 18
Priere de saisir un entier:50
Vous avez lu une valeur entiere egale a 50

-----
Process exited after 10.73 seconds with return value 0
Appuyez sur une touche pour continuer...
```

Figure 1: création de deux threads concurrents

Nous pouvons ajouter d'autres threads dans le programme pour afficher la même valeur lue en hexadécimal et en octal, nous pouvons aussi modifier le programme pour qu'il affiche le temps pris par chaque opération; la lecture, l'affichage et aussi le temps de réponse depuis son lancement jusqu'à sa terminaison.

```

Bienvenue chez le thread ayant comme argument 9
Priere de saisir un entier:12
Temps pour la lecture : 4028
Vous avez lu une valeur entiere egale a : 12
Vous avez lu une valeur octal egale a : 14
Vous avez lu une valeur hexadecimal egale a : c
Temps pour l'affichage' : 1

-----
Process exited after 4.093 seconds with return value 0
Appuyez sur une touche pour continuer...

```

Figure 2: Temps d'exécution du programme

### 3 Synchronisation des pthreads

Dans l'exemple suivant, nous allons exécuter un programme qui permet d'illustrer la synchronisation entre les threads en utilisant des mutexes et des variables de conditions. Le but du programme est de gérer des transactions bancaires sur un compte à condition que le solde du compte ne descende jamais au-dessous de zéro. La figure ci-dessous nous montre que le solde initial du compte bancaire est 200dh. Ensuite, chaque client prend une somme d'argent qui est générée aléatoirement entre 1 et 50 dirhams. Si la valeur du solde bancaire est insuffisante pour débiter la valeur générée, on le crédite d'un montant de 200 dirhams.

```

Creation du thread de la banque!
Creation des threads clients !
Client 1 prend 0 dirhams, reste 200 en solde!
Client 2 prend 0 dirhams, reste 200 en solde!
Client 0 prend 0 dirhams, reste 200 en solde!
Client 4 prend 0 dirhams, reste 200 en solde!
Client 3 prend 0 dirhams, reste 200 en solde!
Client 1 prend 11 dirhams, reste 189 en solde!
Client 2 prend 11 dirhams, reste 178 en solde!
Client 0 prend 11 dirhams, reste 167 en solde!
Client 3 prend 11 dirhams, reste 156 en solde!
Client 4 prend 11 dirhams, reste 145 en solde!
Client 1 prend 35 dirhams, reste 110 en solde!
Client 2 prend 35 dirhams, reste 75 en solde!
Client 0 prend 35 dirhams, reste 40 en solde!
Client 4 prend 35 dirhams, reste 5 en solde!
      Alimentation du compte bancaire avec 200 dirhams!
Client 3 prend 35 dirhams, reste 165 en solde!
Client 1 prend 21 dirhams, reste 144 en solde!
Client 2 prend 21 dirhams, reste 123 en solde!
Client 0 prend 21 dirhams, reste 102 en solde!
Client 4 prend 21 dirhams, reste 81 en solde!
Client 3 prend 21 dirhams, reste 60 en solde!

```

Figure 3: gestion des transactions bancaires

Chaque thread demande la permission d'entrer dans la zone critique par verrouiller le mutex afin qu'il puisse accéder à la Bank, si la valeur à débiter est disponible, l'opération s'effectue correctement et il laisse le verrou à un autre thread. Si non, il faut réveiller le thread du Bank qui est bloqué sur une variable de condition. Après avoir alimenté le solde, le thread de Bank génère une notification pour qu'il permet au thread client de continuer son opération qui sert à prendre la somme d'argent qu'il veut.

Si on n'utilise pas les variables de condition, la probabilité pour que le verrou soit occupé par un thread client est très petite, alors le solde va être presque toujours égal à 200dhs.

```

Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Client 0 prend 10 dirhams, reste 190 en solde!
Client 3 prend 10 dirhams, reste 180 en solde!
Client 4 prend 10 dirhams, reste 170 en solde!
Client 2 prend 10 dirhams, reste 160 en solde!
Client 1 prend 10 dirhams, reste 150 en solde!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!

```

Figure 4: Cas où on supprime les variables de conditions dans le programme

Lorsqu'on supprime les mutexes, le solde de compte bancaire après chaque transaction peut être erroné ; car plus de deux threads concurrents essaient en même temps de modifier une variable globale qui est dans notre cas le solde, un thread modifie le solde bancaire tandis qu'un autre thread essaie de le lire.

```

Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!
Client 4 prend 35 dirhams, reste 130 en solde!
Client 1 prend 35 dirhams, reste 95 en solde!
Client 2 prend 35 dirhams, reste 60 en solde!
Client 3 prend 35 dirhams, reste 25 en solde!
Client 0 prend 35 dirhams, reste 165 en solde!
Alimentation du compte bancaire avec 200 dirhams!
Alimentation du compte bancaire avec 200 dirhams!

```

Figure 5: Cas où on supprime les mutexes dans le programme

Maintenant, nous allons modifier notre programme pour que nous puissions mettre une limite à notre solde bancaire pour fixer le plafond de ces dépenses à 1000 dirhams. Ensuite, nous exécutons le programme :

```
Client 0 prend 30 dirhams, reste 155 en solde!
Client 1 prend 4 dirhams, reste 151 en solde!
Client 1 prend 7 dirhams, reste 144 en solde!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
      impossible de prend 1 dirhams, la limite de 1000dh par jour!
Client 1 prend 5 dirhams, reste 139 en solde!
Client 0 prend 8 dirhams, reste 131 en solde!
Client 1 prend 0 dirhams, reste 131 en solde!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
      impossible de prend 0 dirhams, la limite de 1000dh par jour!
      impossible de prend 1 dirhams, la limite de 1000dh par jour!
Client 2 prend 8 dirhams, reste 123 en solde!
      impossible de prend 0 dirhams, la limite de 1000dh par jour!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
      impossible de prend 1 dirhams, la limite de 1000dh par jour!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
```

Figure 6: Mettre en place le plafond de 100dh

Après l'observation du temps pris par chaque transaction, on constate que si le solde est insuffisant, la transaction prend plus de temps que celui pris par une simple transaction.

```
Client 0 prend 30 dirhams, reste 155 en solde!
Client 1 prend 4 dirhams, reste 151 en solde!
Client 1 prend 7 dirhams, reste 144 en solde!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
      impossible de prend 1 dirhams, la limite de 1000dh par jour!
Client 1 prend 5 dirhams, reste 139 en solde!
Client 0 prend 8 dirhams, reste 131 en solde!
Client 1 prend 0 dirhams, reste 131 en solde!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
      impossible de prend 0 dirhams, la limite de 1000dh par jour!
      impossible de prend 1 dirhams, la limite de 1000dh par jour!
Client 2 prend 8 dirhams, reste 123 en solde!
      impossible de prend 0 dirhams, la limite de 1000dh par jour!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
      impossible de prend 1 dirhams, la limite de 1000dh par jour!
      impossible de prend 2 dirhams, la limite de 1000dh par jour!
```

Figure 7: Mettre en place le plafond de 100dh

## 4 *Communication entre les pthreads*

Dans ce qui suit, nous allons donner un aperçu général sur les principaux mécanismes qu'on peut utiliser pour la communication entre les threads, en se focalisant sur la file de messages, ce mécanisme MQ fonctionne comme un service de gestion de files. Les files contiennent des messages avec des priorités variables, et les messages les plus prioritaires et les plus anciens sont lus en premier.

Ci-dessous une illustration de ce mécanisme. Il s'agit d'une petite application à deux threads communicants (un émetteur et un récepteur) pour calculer le temps nécessaire pour la transition des messages entre les deux parties.

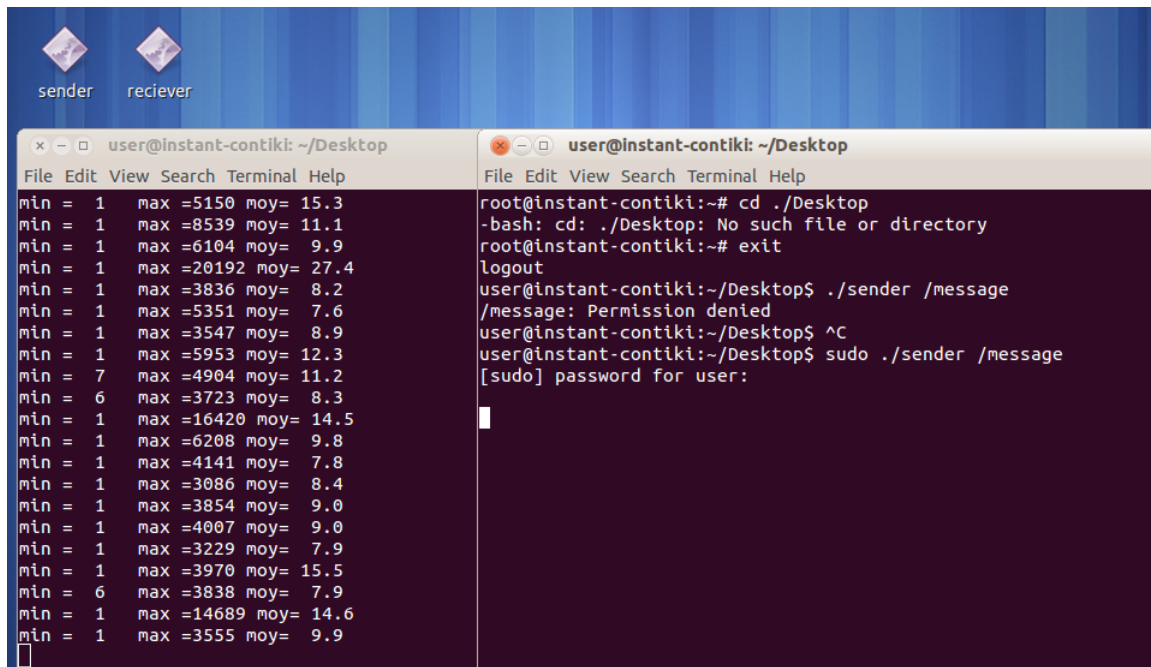


Figure 8: Communication entre un émetteur et un récepteur

L'émetteur lit l'heure sur son système et l'encapsule dans un message, puis l'écrit dans la file d'attente des messages pour le destinataire. Le destinataire de son tour, lit le message dans la file d'attente et lit également l'heure sur son système, puis calcule la différence entre ces deux quantités pour estimer le temps minimum, le temps maximum et le temps moyen passé pendant la communication.

## 5 Étude de cas

### 5.1 Situation

Nous voulons développer une application permettant de traiter des données massives à travers une grande matrice carrée ( $n \times n$ ) dont les valeurs représentent des températures collectées en temps réel en interrogeant régulièrement, pendant une période donnée, des capteurs déployés dans une zone géographique sensible. L'objectif est de calculer en temps réel des statistiques (température moyenne, température médiane, l'écart-type et la variance de températures), et de les afficher sous forme d'un tableau de bord pour une analyse et une prise de décisions par le manager. Les résultats vont être calculés en utilisant le multithreading.

### 5.2 L'architecture de l'application

Nous allons déployer plusieurs capteurs dans la zone géographique sensible. Ces derniers vont mesurer régulièrement la température et l'envoyer un à un à un sink ou une station de base qui va se charger de calculer la température moyenne, la température médiane, l'écart-type et la variance de températures. Ces statistiques calculées seront envoyées ensuite via internet au manager qui va les visualiser dans un tableau de bord pour les étudier et prendre les décisions.

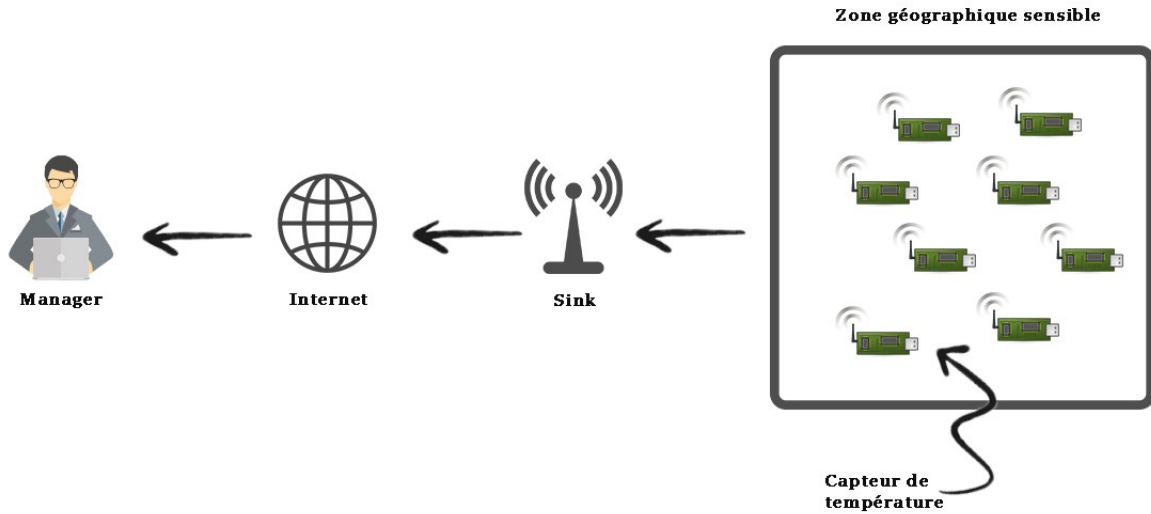


Figure 9: Architecture physique et déploiement des capteurs

### 5.3 Exécution du code

Nous exécutons le code fourni en faisant changer le nombre de threads à utiliser tout en calculant à chaque fois le temps d'exécution de l'application et le pourcentage d'utilisation de temps du processeur.

```
C:\Users\achraf\Desktop\etude\Nouary2\posix\tp3p41.exe
Your matrix is as follows:
[0:20:40:60:80:100:120:140:160:180:200:220:240:260:280:300:320:340:360:380:]
[1:21:41:61:81:101:121:141:161:181:201:221:241:261:281:301:321:341:361:381:]
[2:22:42:62:82:102:122:142:162:182:202:222:242:262:282:302:322:342:362:382:]
[3:23:43:63:83:103:123:143:163:183:203:223:243:263:283:303:323:343:363:383:]
[4:24:44:64:84:104:124:144:164:184:204:224:244:264:284:304:324:344:364:384:]
[5:25:45:65:85:105:125:145:165:185:205:225:245:265:285:305:325:345:365:385:]
[6:26:46:66:86:106:126:146:166:186:206:226:246:266:286:306:326:346:366:386:]
[7:27:47:67:87:107:127:147:167:187:207:227:247:267:287:307:327:347:367:387:]
[8:28:48:68:88:108:128:148:168:188:208:228:248:268:288:308:328:348:368:388:]
[9:29:49:69:89:109:129:149:169:189:209:229:249:269:289:309:329:349:369:389:]
[10:30:50:70:90:110:130:150:170:190:210:230:250:270:290:310:330:350:370:390:]
[11:31:51:71:91:111:131:151:171:191:211:231:251:271:291:311:331:351:371:391:]
[12:32:52:72:92:112:132:152:172:192:212:232:252:272:292:312:332:352:372:392:]
[13:33:53:73:93:113:133:153:173:193:213:233:253:273:293:313:333:353:373:393:]
[14:34:54:74:94:114:134:154:174:194:214:234:254:274:294:314:334:354:374:394:]
[15:35:55:75:95:115:135:155:175:195:215:235:255:275:295:315:335:355:375:395:]
[16:36:56:76:96:116:136:156:176:196:216:236:256:276:296:316:336:356:376:396:]
[17:37:57:77:97:117:137:157:177:197:217:237:257:277:297:317:337:357:377:397:]
[18:38:58:78:98:118:138:158:178:198:218:238:258:278:298:318:338:358:378:398:]
[19:39:59:79:99:119:139:159:179:199:219:239:259:279:299:319:339:359:379:399:]
Please hit enter to continue ...
Thread 0 just terminated with its partial sum 3800
Thread 1 just terminated with its partial sum 3820
Thread 2 just terminated with its partial sum 3840
Thread 3 just terminated with its partial sum 3860
Thread 4 just terminated with its partial sum 3880
Thread 5 just terminated with its partial sum 3900
Thread 6 just terminated with its partial sum 3920
Thread 7 just terminated with its partial sum 3940
Thread 8 just terminated with its partial sum 3960
Thread 9 just terminated with its partial sum 3980
Thread 10 just terminated with its partial sum 4000
Thread 11 just terminated with its partial sum 4020
Thread 12 just terminated with its partial sum 4040
Thread 13 just terminated with its partial sum 4060
Thread 14 just terminated with its partial sum 4080
Thread 15 just terminated with its partial sum 4100
Thread 16 just terminated with its partial sum 4120
Thread 17 just terminated with its partial sum 4140
Thread 18 just terminated with its partial sum 4160
Thread 19 just terminated with its partial sum 4180
The total sum calculated by all the threads is 79800
1.367000
-----
Process exited after 1.467 seconds with return value 0
```

Figure 10: Exécution du code



Nous remarquons que le temps d'exécution du programme devient plus long lorsque nous avons plus de threads.

Après l'exécution du programme, nous constatons qu'une nouvelle application apparaît dans le gestionnaire de tâches (console Pauser.exe). Cette dernière a ajouté 0,8% d'utilisation au processeur. Nous pouvons donc dire que le programme utilise entre 0 et 0,8% du processeur.

## **6 *Conclusion***

Ce travail a été très bénéfique pour apprendre le développement en temps réel avec POSIX à travers la compréhension des threads POSIX, la synchronisation des pthreads, la communication entre les pthreads et enfin un étude de cas.

Voici le lien où figure le code source du travail : <https://github.com/elganiesta/POSIX>