Semaphores

Parallel and Distributed Systems 2019-2020

AMIARD Landry

We suppose that NB_CONS is the number of consumers, NB_PROD the number of consumers NB_POS the size of the buffer and NB_TYPE the number of message types.

Version 1:

```
Semaphore prodSem, consSem ;
Mutex m ;
init (prodSem, NB_POS) ;
init (consSem, 0) ;
init (m, 1) ;

Producer1 (message msg) {
        P(prodSem) ;
        m.lock ;
        insert(buffer, posP, msg) ;
        posP = (posP + 1)%NB_POS ;
        m.unlock ;
        V(consSem) ;
}
Consumer1 (message msg) {
        P(consSem) ;
        m.lock ;
        extract(buffer, posC, msg) ;
        posC = (posC + 1)%NB_POS ;
        m.unlock ;
        V(prodSem) ;
}
```

Version 2:

```
Semaphore prod0Sem, prod1Sem, prodEntrySem, consSem ;
Mutex m ;
init (prodEntrySem, NB_POS) ; init (prod0Sem, 1) ;
init (prod1Sem, 0) ; init (consSem, 0) ; init (m, 1) ;

Producer2 (message msg) {
        P(prodEntrySem) ;
        if(msg.type = 0)
                P(prod0Sem) ;
        if(msg.type = 1)
                P(prod1Sem) ;
        m.lock ;
        insert(buffer, posP, msg) ;
        posP = (posP + 1)%NB_POS ;
        m.unlock ;
        if (msg.type = 0)
                V(prod1Sem) ;
        if (msg.type = 1)
                V(prod0Sem) ;
        V(consSem) ;
}
Consumer2 (message msg) {
        P(consSem) ;
        m.lock ;
        extract(buffer, posC, msg) ;
        posC = (posC + 1)%NB_POS ;
        m.unlock ;
        V(prodEntrySem) ;
}
```

Version 3:

```
Semaphore prodSem, consEntrySem ;
Semaphore consSem[NB_TYPE] ;
Mutex m ;
for (int i = 0 ; i < NB_TYPE ; i++)
        init (consSem[i],  0) ;
init (prodSem, NB_POS) ; init (consEntrySem, 0) ;
init (m, 1) ;

Producer3 (message msg) {
        P(prodSem) ;
        m.lock ;
        insert(buffer, posP, msg) ;
        posP = (posP + 1)%NB_POS ;
        m.unlock ;
        V(consEntrySem) ;
}

Consumer3 (message msg) {
        P(consEntrySem) ;
        for (int i = 0 ; i < NB_TYPE ; i++)
                if(msg.type = i)
                        P(consSem[i]) ;
        m.lock ;
        extract(buffer, posP, msg) ;
        m.unlock ;
        V(prodSem) ;
        //This version is not working, I wasn't able to find a solution to the problem of choosing the good consumer to wake up (regarding the message's type).
}
```

Version 1, C code :

* Modfied code is in blue *

```
//Global variables :

pthread_mutex_t criticalMutex = PTHREAD_MUTEX_INITIALIZER;
sem_t producerSem ;
sem_t consumerSem ;
```

```
void * consumer (void *arg) {
  int i;
  TypeMessage theMessage;
  Parameters *param = (Parameters *)arg;
            sleep(1);
  for (i = 0; i < NB_TIMES_CONS; i++) {
              sem_wait(&consumerSem);
              pthread_mutex_lock(&criticalMutex);
              makeGet(&theMessage);
              sem_post(&producerSem);
              pthread_mutex_unlock(&criticalMutex);
  }
  return NULL;
}
```

```
void * producer (void *arg) {
  int i;
  TypeMessage theMessage;
  Parameters *param = (Parameters *)arg;
            sleep(1);
  for (i = 0; i < NB_TIMES_PROD; i++) {
    theMessage.typeOfMessage = param->typeOfMessage;
    theMessage.producerNumber = param->threadNumber;
              sem_wait(&producerSem);
              pthread_mutex_lock(&criticalMutex);
              makePut(theMessage);
              sem_post(&consumerSem);
              pthread_mutex_unlock(&criticalMutex);
  }
  return NULL;
}
```

```
//main() function :

initializeSharedVariables();
sem_init(&producerSem, 0, nbPositions);
sem_init(&consumerSem, 0, 0);

for (i = 0; i <  nbThds; i++) {
  if (i < nbProd) {
    paramThds[i].typeOfMessage = 0;
    paramThds[i].threadNumber = i;
    if ((etat = pthread_create(&idThdProd[i], NULL, producer, &paramThds[i])) != 0)
      thdErreur(etat, "Creation producer", etat);
  }
  else {
    paramThds[i].typeOfMessage = 0;
    paramThds[i].threadNumber = i - nbProd;
    if ((etat = pthread_create(&idThdConso[i-nbProd], NULL, consumer, &paramThds[i])) != 0)
      thdErreur(etat, "Creation consumer", etat);
  }
}
```

Version 2, C code :

* Modfied code is in blue *

```
//Global variables :

pthread_mutex_t criticalMutex = PTHREAD_MUTEX_INITIALIZER;
sem_t producerEntrySem ;
sem_t producer0Sem ;
sem_t producer1Sem ;
sem_t consumerSem ;
```

```
void * consumer (void *arg) {
 int i;
 TypeMessage theMessage;
 Parameters *param = (Parameters *)arg;
         sleep(1);
 for (i = 0; i < NB_TIMES_CONS; i++) {
          sem_wait(&consumerSem);
          pthread_mutex_lock(&criticalMutex);

          makeGet(&theMessage);

          sem_post(&producerEntrySem);
          pthread_mutex_unlock(&criticalMutex);
 }
 return NULL;
}
```

```
void * producer (void *arg) {
 int i;
 TypeMessage theMessage;
 Parameters *param = (Parameters *)arg;
         sleep(1);
 for (i = 0; i < NB_TIMES_PROD; i++) {
  theMessage.typeOfMessage = param->typeOfMessage;
  theMessage.producerNumber = param->threadNumber;
  sem_wait(&producerEntrySem);
  if (theMessage.typeOfMessage==0) {
   sem_wait(&producer0Sem);
  }
  if (theMessage.typeOfMessage==1) {
   sem_wait(&producer1Sem);
  }

  pthread_mutex_lock(&criticalMutex);
  makePut(theMessage);
  pthread_mutex_unlock(&criticalMutex);

  if (theMessage.typeOfMessage==0) {
   sem_post(&producer1Sem);
  }
  if (theMessage.typeOfMessage==1) {
   sem_post(&producer0Sem);
  }
  sem_post(&consumerSem);
 }
 return NULL;
}
```

```
//main() function

initializeSharedVariables();
sem_init(&producerEntrySem, 0, nbPositions);
sem_init(&producer0Sem, 0, 1);
sem_init(&producer1Sem, 0, 0);
sem_init(&consumerSem, 0, 0);

for (i = 0; i <  nbThds; i++) {
 if (i < nbProd) {
  paramThds[i].typeOfMessage = i%2;
  paramThds[i].threadNumber = i;
  if ((etat = pthread_create(&idThdProd[i], NULL, producer, &paramThds[i])) != 0)
   thdErreur(etat, "Creation producer", etat);
 }
 Else {
  //consumers consume messages of any types so we don't add a type of message
  paramThds[i].threadNumber = i - nbProd;
  if ((etat = pthread_create(&idThdConso[i-nbProd], NULL, consumer, &paramThds[i])) != 0)
   thdErreur(etat, "Creation consumer", etat);
 }
}
```

Version 3, C code :

* Modfied code is in blue *

```
//Global Variables :

#define NB_TYPES 5
pthread_mutex_t criticalMutex = PTHREAD_MUTEX_INITIALIZER;
sem_t producerSem ;
sem_t consumerEntrySem ;
sem_t consumerSem[NB_TYPES] ;
```

```
void * producer (void *arg) {
  int i;
  TypeMessage theMessage;
  Parameters *param = (Parameters *)arg;
  sleep(1);
  theMessage.typeOfMessage = param->typeOfMessage;
  theMessage.producerNumber = param→threadNumber;
  for (i = 0; i < NB_TIMES_PROD; i++) {
            sem_wait(&producerSem);
            pthread_mutex_lock(&criticalMutex);
            makePut(theMessage);
            pthread_mutex_unlock(&criticalMutex);
            sem_post(&consumerEntrySem]);
  }
  return NULL;
}
```

```
void * consumer (void *arg) {
  int i;
  TypeMessage theMessage;
  Parameters *param = (Parameters *)arg;
            sleep(1);
  for (i = 0; i < NB_TIMES_CONS; i++) {
    sem_wait(&consumerEntrySem);
    for (int i = 0 ; i < NB_TYPE ; i++){
       if(theMessage.typeOfMessage == i) {
         sem_post(&consSem[i]) ;
        }
    }
    pthread_mutex_lock(&criticalMutex);
    makeGet(&theMessage);
    sem_post(&producerSem);
    pthread_mutex_unlock(&criticalMutex);
  }
  return NULL;
}
```

```
//main() fucntion :

initializeSharedVariables();
sem_init(&producerSem, 0, nbPositions);
for (int i = 0 ; i < NB_TYPES ; i++){
  sem_init(&consumerSem[i], 0, 0);
}

for (i = 0; i <  nbThds; i++) {
  if (i < nbProd) {
    paramThds[i].typeOfMessage = i%NB_TYPES;
    paramThds[i].threadNumber = i;
    if ((etat = pthread_create(&idThdProd[i], NULL, producer, &paramThds[i])) != 0)
      thdErreur(etat, "Creation producer", etat);
  }
  else {
    paramThds[i].typeOfMessage = i%NB_TYPES;
    paramThds[i].threadNumber = i - nbProd;
    if ((etat = pthread_create(&idThdConso[i-nbProd], NULL, consumer, &paramThds[i])) !=
0)
      thdErreur(etat, "Creation consumer", etat);
  }
}
```