Universidad Autónoma Metropolitana
Unidad Azcapotzalco

Sistemas Operativos

Práctica 04: Comunicación entre procesos.

1. Introducción

1.1. Condiciones de carrera

Estas condiciones se generan en situaciones en las que dos o más procesos leen o escriben datos compartidos y el resultado final depende de el orden en cual se ejecuten estos procesos. En la depuración de programas que contienen condiciones de carrera los resultados de la mayor parte de las pruebas son correctos, pero de vez en cuando algo raro e inexplicable sucede.

1.2. Exclusión mutua

Para evitar las condiciones de carrera y otras situaciones en las que se comparte memoria, archivos o cualquier otro recurso compartido se debe implementar la exclusión mutua. Las regiones o secciones críticas son aquellas en las que un proceso está accediendo a un recurso compartido o efectuando otras tareas críticas que pueden dar lugar a competencias. Cuando se organizan las cosas de modo que dos procesos nunca pudieran estar en sus regiones críticas al mismo tiempo, se pueden evitar las condiciones de carrera. Para tener una buena solución se requiere que se cumplan las siguientes condiciones:

- ✓ Dos procesos nunca pueden estar simultáneamente dentro de sus regiones críticas.
- ✓ No puede suponerse nada acerca de las velocidades o el número de las CPU.
- ✓ Ningún proceso que se ejecute fuera de su región crítica puede bloquear a otros procesos.
- ✓ Ningún proceso deberá tener que esperar indefinidamente para entrar a su región crítica.

2. Códigos y ejecución

2.1. Problema del productor consumidor

```
#include<stdio.h>
#include<pthread.h>

#define N 10
#define MAX 10

char bufer[N];
```

```
pthread_mutex_t m;
pthread_cond_t condc, condp;
int pos = 0;

void imprime_buf() {
  int i;
  for (i = 0; i < N; ++ i) {
    printf("%c ", bufer[i]);
  }
}
void *productor(void*ptr) {
  int i;
  for (i = 1; i < MAX; ++i) {
    pthread_mutex_lock (&m);
    while (pos != 0) {
      pthread_cond_wait (&condp, &m);
    }
    bufer[pos] = 'p';
    pos = pos + 1;
    imprime_buf();
    pthread_cond_signal(&condc);
    pthread_mutex_unlock(&m);
  }
  pthread_exit(0);
}

void *consumidor(void *ptr) {
  int i;
  for (i = 1; i < MAX; ++i) {
    pthread_mutex_lock(&m);
    while(pos == 0) {
      pthread_cond_wait (&condc,&m);
    }
    pos = pos - 1;
    bufer[pos] = NULL;
    imprime_buf();
    pthread_cond_signal(&condp);
    pthread_mutex_unlock(&m);
  }
  pthread_exit(0);
}
int main(int argc, char ** argv) {
  pthread_t pro, con;
  pthread_mutex_init(&m, 0);
  pthread_cond_init(&condc, 0);
  pthread_cond_init(&condp, 0);
  pthread_create(&con, NULL, consumidor, NULL);
  pthread_create(&pro, NULL, productor, NULL);
  pthread_join(pro, NULL);
```

```
  pthread_join(con, NULL);
  pthread_cond_destroy(&condc);
  pthread_cond_destroy(&condp);
  pthread_mutex_destroy(&m);
  return 0;

}
```

Al ejecutar el código anterior , su salida es la siguiente:

```
[rebeca@n00 exlu]$ gcc e_m1.c -o e_m1 -lpthread
e_m1.c: En la funci n     consumidor    :
e_m1.c:46:14: aviso: la asignaci n crea un entero desde un puntero
sin una conversi n [-Wint-conversion]
   bufer[pos] = NULL;
              ^
[rebeca@n00 exlu]$ ./e_m1
p                   p                   p
  p                   p                   p
    p                   p                   p
[rebeca@n00 exlu]$
```

2.2. Productor genera 5 productos

```
#include<stdio.h>
#include<pthread.h>

#define N 10
#define MAX 10

char bufer[N];
pthread_mutex_t m;
pthread_cond_t condc, condp;
int pos = 0;

void imprime_buf() {
  int i;
  for (i = 0; i < N; ++ i) {
    printf("%c ", bufer[i]);
  }
}
void *productor(void*ptr) {
  int i;
  for (i = 1; i < MAX; ++i) {
    pthread_mutex_lock (&m);
    if (pos == 5) {
      pthread_cond_wait (&condp, &m);
    }
    if (pos == 0) {
```

```
      for (int j = 0; j < 5; ++j) {
        bufer[pos] = 'p';
        pos = pos + 1;
        imprime_buf();
      }
    }
    pthread_cond_signal(&condc);
    pthread_mutex_unlock(&m);

  }
  pthread_exit(0);
}

void *consumidor(void *ptr) {
  int i;
  for (i = 1; i < MAX; ++i) {
    pthread_mutex_lock(&m);
    if (pos == 0) {
      pthread_cond_wait (&condc,&m);
    }
    if (pos != 0) {
      pos = pos - 1;
      bufer[pos] = NULL;
      imprime_buf();
      pthread_cond_signal(&condp);
      pthread_mutex_unlock(&m);
    }
  }
  pthread_exit(0);
}
int main(int argc, char ** argv) {
  pthread_t pro, con;
  pthread_mutex_init(&m, 0);
  pthread_cond_init(&condc, 0);
  pthread_cond_init(&condp, 0);
  pthread_create(&con, NULL, consumidor, NULL);
  pthread_create(&pro, NULL, productor, NULL);
  pthread_join(pro, NULL);
  pthread_join(con, NULL);
  pthread_cond_destroy(&condc);
  pthread_cond_destroy(&condp);
  pthread_mutex_destroy(&m);
  return 0;
}
```

Al ejecutar el código anterior , su salida es la siguiente:

```
[rebeca@n00 exlu]$ gcc e_m2.c -o e_m2 -lpthread
e_m2.c: En la funci n     consumidor    :
e_m2.c:48:15: aviso: la asignaci n crea un entero desde un puntero
sin una conversi n [-Wint-conversion]
    bufer[pos] = NULL;
                 ^
[rebeca@n00 exlu]$ ./e_m2
p           p p           p p p         p p p p       p p p p p
p p p p       p p p         p p           p
p           p p           p p p         p p p p       p p p p p
p p p p       p p p         p p           p
```

2.3. Consumidor toma 5 productos

```
#include<stdio.h>
#include<pthread.h>

#define N 10
#define MAX 10

char bufer[N];
pthread_mutex_t m;
pthread_cond_t condc, condp;
int pos = 0;

void imprime_buf() {
  int i;
  for (i = 0; i < N; ++ i) {
    printf("%c ", bufer[i]);
  }
}
void *productor(void*ptr) {
  int i;
  for (i = 1; i < MAX; ++i) {
    pthread_mutex_lock (&m);
    if (pos == MAX) {
      pthread_cond_wait (&condp, &m);
    }
    if (pos < MAX) {
      bufer[pos] = 'p';
      pos = pos + 1;
      imprime_buf();
      pthread_cond_signal(&condc);
      pthread_mutex_unlock(&m);
    }
```

```
  }
  pthread_exit(0);
}

void *consumidor(void *ptr) {
  int i, j;
  for (i = 1; i < MAX; ++i) {
    pthread_mutex_lock(&m);
    while (pos == 0) {
      pthread_cond_wait (&condc,&m);
    }
    if (pos >= 5) {
      for (j = 0; j < 5; ++j){
        pos = pos - 1;
        bufer[pos] = NULL;
        imprime_buf();
      }
    }
    pthread_cond_signal(&condp);
    pthread_mutex_unlock(&m);
  }
  pthread_exit(0);
}
int main(int argc, char ** argv) {
  pthread_t pro, con;
  pthread_mutex_init(&m, 0);
  pthread_cond_init(&condc, 0);
  pthread_cond_init(&condp, 0);
  pthread_create(&pro, NULL, productor, NULL);
  pthread_create(&con, NULL, consumidor, NULL);
  pthread_join(pro, NULL);
  pthread_join(con, NULL);
  pthread_cond_destroy(&condc);
  pthread_cond_destroy(&condp);
  pthread_mutex_destroy(&m);
  return 0;
}
```

Al ejecutar el código anterior , su salida es la siguiente:

```
[rebeca@n00 exlu]$ gcc e_m3.c -o e_m3 -lpthread
e_m3.c: En la funci n      consumidor     :
e_m3.c:47:16: aviso: la asignaci n crea un entero desde un puntero
sin una conversi n [-Wint-conversion]
      bufer[pos] = NULL;
                   ^
[rebeca@n00 exlu]$ ./e_m3
p           p p           p p p          p p p p        p p p p p
p p p p p p      p p p p p p p p     p p p p p p p p
p p p p p p p p p   p p p p p p p p   p p p p p p p p    p p p p p p
p p p p p       p p p p
```

3. Cuestionario

1. ¿Qué pasa si tanto el productor como el consumidor despiertan a su contraparte una vez que han salido de su región crítica?¿Cuál es la diferencia con respecto a no hacerlo?
Cuando tanto el productor y consumidor despiertan a su contraparte una vez que han salido de su región crítica sucede que en cuanto se genera un producto por el productor, el consumidor lo toma, despierta al productor y este vuelve a generar un producto. La diferencia depende de la condición que se genere para que el hilo despierte a su contraparte.
2. ¿En qué momento se generan condiciones de carrea en su programa?
En el momento que las funciones ejecutadas por hilos diferentes quieren acceder al recurso compartido, en este caso es la cadena $bufer[]$y el entero pos.
3. Indique las regiones críticas en su código e inclúyalas en su reporte. Justifique su respuesta
Tanto en la función $productor()$ como en la funcion $consumidor()$, cuando se accede a las variables globales $bufer[]$ y pos.
En la función productor:

```
bufer[pos] = 'p';
pos = pos + 1;
```

En la función consumidor:

```
bufer[pos] = NULL;
pos = pos - 1;
```

4. ¿Qué solución implementó para garantizar exclusión mutua?
Con variables tipo mutex.