ITP 30002 Operating System

Conditional Variable & Semaphore

OSTEP Chapters 30 and 31

Shin Hong

Motivation

- There are many times where a thread wishes to check whether a condition is satisfied before continuing its execution

- Example: thread join

```
1   volatile int done = 0;
2
3   void *child(void *arg) {
4       printf("child\n");
5       done = 1;
6       return NULL;
7   }
8
9   int main(int argc, char *argv[]) {
10      printf("parent: begin\n");
11      pthread_t c;
12      Pthread_create(&c, NULL, child, NULL); // create child
13      while (done == 0)
14          ; // spin
15      printf("parent: end\n");
16      return 0;
17  }
```

Conditional
Variables &
Semaphores

ITP 30002
Operating System

2021-05-29

Conditional Variable

- A conditional variable is an explicit queue that threads can put themselves on when some state of execution is not desired
  - when it said the condition might be changed, a waiting thread is awken up to continue its execution
  - operation
    - wait () `pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m);`
    - signal () `pthread_cond_signal(pthread_cond_t *c);`

- a conditional variable is used together with a mutex for ensuring mutual exclusion of the condition checking

Example

```
1    int done  = 0;
2    pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
3    pthread_cond_t  c  = PTHREAD_COND_INITIALIZER;
4
5    void thr_exit() {
6        Pthread_mutex_lock(&m);
7        done = 1;
8        Pthread_cond_signal(&c);
9        Pthread_mutex_unlock(&m);
10   }
11
12   void *child(void *arg) {
13       printf("child\n");
14       thr_exit();
15       return NULL;
16   }
17
18   void thr_join() {
19       Pthread_mutex_lock(&m);
20       while (done == 0)
21           Pthread_cond_wait(&c, &m);
22       Pthread_mutex_unlock(&m);
23   }
24
25   int main(int argc, char *argv[]) {
26       printf("parent: begin\n");
27       pthread_t p;
28       Pthread_create(&p, NULL, child, NULL);
29       thr_join();
30       printf("parent: end\n");
31       return 0;
32   }
```

Conditional
Variables &
Semaphores

ITP 30002
Operating System

2021-05-29

Producer Consumer Problem

- There are one or more **producer threads** each of which place a task with data item at a buffer, and one or more **consumer threads** that takes a placed task and process it one by one

- Requirements
  - a producer must be blocked if the buffer is full
  - a consumer must be blocked if there is no task found in buffer
  - accesses to a shared buffer must be synchronized

```
1   int buffer[MAX];
2   int fill_ptr = 0;
3   int use_ptr  = 0;
4   int count    = 0;
5
6   void put(int value) {
7       buffer[fill_ptr] = value;
8       fill_ptr = (fill_ptr + 1) % MAX;
9       count++;
10  }
11
12  int get() {
13      int tmp = buffer[use_ptr];
14      use_ptr = (use_ptr + 1) % MAX;
15      count--;
16      return tmp;
17  }
1   void *producer(void *arg) {
2       int i;
3       int loops = (int) arg;
4       for (i = 0; i < loops; i++) {
5           put(i);
6       }
7   }
8
9   void *consumer(void *arg) {
10      while (1) {
11          int tmp = get();
12          printf("%d\n", tmp);
13      }
14  }
```

```
1   cond_t empty, fill;
2   mutex_t mutex;
3
4   void *producer(void *arg) {
5       int i;
6       for (i = 0; i < loops; i++) {
7           Pthread_mutex_lock(&mutex);
8           while (count == MAX)
9               Pthread_cond_wait(&empty, &mutex);
10          put(i);
11          Pthread_cond_signal(&fill);
12          Pthread_mutex_unlock(&mutex);
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          Pthread_mutex_lock(&mutex);
20          while (count == 0)
21              Pthread_cond_wait(&fill, &mutex);
22          int tmp = get();
23          Pthread_cond_signal(&empty);
24          Pthread_mutex_unlock(&mutex);
25          printf("%d\n", tmp);
26      }
27  }
```

Semaphore

- A counter-like synchronization primitive
  - represents the number of available shared resources
  - provides increase and decrease operations
    - decrease the counter when a thread is checking out a resource
      - block the caller thread if the counter became negative
    - increase the counter when a thread is putting back a held resource
      - wake up a blocked thread if the counter became zero

- POSIX semaphore
  - sem_init ()
  - sem_wait (), sem_post ()

Semaphore As Mutex

```
1    sem_t m;
2    sem_init(&m, 0, 1 );
3
4    sem_wait(&m);
5    // critical section here
6    sem_post(&m);
```

Val	Thread 0	State	Thread 1	State
1		Run		Ready
1	call sem_wait()	Run		Ready
0	sem_wait() returns	Run		Ready
0	(crit sect begin)	Run		Ready
0	*Interrupt; Switch→T1*	Ready		Run
0		Ready	call sem_wait()	Run
-1		Ready	decr sem	Run
-1		Ready	(sem<0)→sleep	Sleep
-1		Run	*Switch→T0*	Sleep
-1	(crit sect end)	Run		Sleep
-1	call sem_post()	Run		Sleep
0	incr sem	Run		Sleep
0	wake(T1)	Run		Ready
0	sem_post() returns	Run		Ready
0	*Interrupt; Switch→T1*	Ready		Run
0		Ready	sem_wait() returns	Run
0		Ready	(crit sect)	Run
0		Ready	call sem_post()	Run
1		Ready	sem_post() returns	Run

Semaphore as Ordering Primitive

```
1   sem_t s;
2
3   void *child(void *arg) {
4       printf("child\n");
5       sem_post(&s); // signal here: child is done
6       return NULL;
7   }
8
9   int main(int argc, char *argv[]) {
10      sem_init(&s, 0, X); // what should X be?
11      printf("parent: begin\n");
12      pthread_t c;
13      Pthread_create(&c, NULL, child, NULL);
14      sem_wait(&s); // wait here for child
15      printf("parent: end\n");
16      return 0;
17  }
```

Conditional
Variables &
Semaphores

ITP 30002
Operating System

2021-05-29

```
1   typedef struct __Zem_t {
2       int value;
3       pthread_cond_t cond;
4       pthread_mutex_t lock;
5   } Zem_t;
```

```
7   // only one thread can call this
8   void Zem_init(Zem_t *s, int value) {
9       s->value = value;
10      Cond_init(&s->cond);
11      Mutex_init(&s->lock);
12  }
```

```
14  void Zem_wait(Zem_t *s) {
15      Mutex_lock(&s->lock);
16      while (s->value <= 0)
17          Cond_wait(&s->cond, &s->lock);
18      s->value--;
19      Mutex_unlock(&s->lock);
20  }
```

```
22  void Zem_post(Zem_t *s) {
23      Mutex_lock(&s->lock);
24      s->value++;
25      Cond_signal(&s->cond);
26      Mutex_unlock(&s->lock);
27  }
```

Producer and Consumer with Semaphore

- Version 1

```
1   sem_t empty;
2   sem_t full;
3
4   void *producer(void *arg) {
5       int i;
6       for (i = 0; i < loops; i++) {
7           sem_wait(&empty);        // Line P1
8           put(i);                  // Line P2
9           sem_post(&full);         // Line P3
10      }
11  }
12
13  void *consumer(void *arg) {
14      int tmp = 0;
15      while (tmp != -1) {
16          sem_wait(&full);         // Line C1
17          tmp = get();             // Line C2
18          sem_post(&empty);        // Line C3
19          printf("%d\n", tmp);
20      }
21  }
22
23  int main(int argc, char *argv[]) {
24      // ...
25      sem_init(&empty, 0, MAX); // MAX are empty
26      sem_init(&full, 0, 0);    // 0 are full
27      // ...
28  }
```

Conditional
Variables &
Semaphores

ITP 30002
Operating System

2021-05-29

Producer and Consumer with Semaphore

- Version 2

```
1    void *producer(void *arg) {
2        int i;
3        for (i = 0; i < loops; i++) {
4            sem_wait(&mutex);        // Line P0 (NEW LINE)
5            sem_wait(&empty);        // Line P1
6            put(i);                  // Line P2
7            sem_post(&full);         // Line P3
8            sem_post(&mutex);        // Line P4 (NEW LINE)
9        }
10   }
11
12   void *consumer(void *arg) {
13       int i;
14       for (i = 0; i < loops; i++) {
15           sem_wait(&mutex);        // Line C0 (NEW LINE)
16           sem_wait(&full);         // Line C1
17           int tmp = get();         // Line C2
18           sem_post(&empty);        // Line C3
19           sem_post(&mutex);        // Line C4 (NEW LINE)
20           printf("%d\n", tmp);
21       }
22   }
```

Producer and Consumer with Semaphore

- Version 3

```
1   void *producer(void *arg) {
2       int i;
3       for (i = 0; i < loops; i++) {
4           sem_wait(&empty);        // Line P1
5           sem_wait(&mutex);        // Line P1.5 (MUTEX HERE)
6           put(i);                  // Line P2
7           sem_post(&mutex);        // Line P2.5 (AND HERE)
8           sem_post(&full);         // Line P3
9       }
10  }
11
12  void *consumer(void *arg) {
13      int i;
14      for (i = 0; i < loops; i++) {
15          sem_wait(&full);         // Line C1
16          sem_wait(&mutex);        // Line C1.5 (MUTEX HERE)
17          int tmp = get();         // Line C2
18          sem_post(&mutex);        // Line C2.5 (AND HERE)
19          sem_post(&empty);        // Line C3
20          printf("%d\n", tmp);
21      }
22  }
```

Conditional Variables & Semaphores

ITP 30002
Operating System

2021-05-29