Systems Programming

Lecture 7:

Inter Process Communication: Semaphores and Messages

Dmitry A. Zaitsev

https://dimazaitsev.github.io

Semaphore

- E.W.Dijkstra, 1962
- Integer variable S
- Initialization S = <number of resources>
- Operations: P – wait, V – signal
- P(S): if(S>0) S=S-1; else <wait on S>
- V(S): if(<there are waiting on S>) <wake up first waiting>; else S=S+1;

Producers - Consumers

emptyCount=N; fullCount=0; useQueue=1;

- Produce:

P(emptyCount)

P(useQueue)

putItemIntoQueue(Item)

V(useQueue)

V(fullCount)

- Consume:

P(fullCount)

P(useQueue)

Item =getItemFromQueue()

V(useQueue)

V(emptyCount)

Linux POSIX semaphores

- #include <fcntl.h> /* For O_* constants */

#include <sys/stat.h> /* For mode constants */

#include <semaphore.h>

- Link -lpthread
- Operations:

P(S): sem_wait(s)

V(S): sem_post(s)

- Name – external, pointer to *sem_t* - internal

Initialize and open a named semaphore

- sem_t *sem_open(const char *name, int oflag);
- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
- The *oflag* argument specifies flags that control the operation of the call: O_CREAT, O_EXCL
- The *mode* argument specifies the permissions to be placed on the new semaphore.
- The *value* argument specifies the initial value for the new semaphore.

Unlock and Lock a semaphore

- Unlock a semaphore

```
int sem_post(sem_t *sem);
```

- Lock a semaphore

```
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
```

Finish using semaphore

- close a named semaphore

```
int sem_close(sem_t *sem);
```

- remove a named semaphore

```
int sem_unlink(const char * name);
```

More functions for semaphores

- get the value of a semaphore

int sem_getvalue(sem_t *sem, int *sval);

- initialize an unnamed semaphore

int sem_init(sem_t *sem, int pshared, unsigned int value);

- destroy an unnamed semaphore

int sem_destroy(sem_t *sem);

A task: dining philosophers

- Pension
- 5 philosophers: thinking or eating
- 5 plates and 5 forks among them
- A philosopher needs 2 forks to start eating
- Deadlock !
- How to avoid deadlocks ?
- Timeouts and other solutions

Messages

- A queue in kernel
- A process sends a message to a queue
- A message is stored in a queue
- If message queue is full, block the sender
- A process receives a message from queue
- If message queue is empty, block the reveiver

POSIX message queues

- #include <mqueue.h>
- Link -lrt
- Name: “/somename”
- Descriptor mqd_t
- Attributes

```
struct   mq_attr   {
    long mq_flags;       /* Flags: 0 or O_NONBLOCK */
    long  mq_maxmsg;      /* Max. # of messages on queue */
    long   mq_msgsize;    /* Max. message size (bytes) */
    long mq_curmsgs;     /* # of messages currently in queue */
};
```

- Basic operations: mq_send(…), mq_receive(…)

Open a message queue

- mqd_t mq_open(const char *name, int oflag);
- mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr);
- The *oflag* argument specifies flags that control the operation of the call: O_RDWR, O_CREAT, O_NONBLOCK
- The *mode* argument specifies the permissions to be placed on the new queue
- The *attr* argument specifies attributes for the queue.

Send a message

- mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio);
- int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio, const struct timespec *abs_timeout);
- struct timespec {

  time_t tv_sec; /* seconds */

  long tv_nsec; /* nanoseconds */

  };

Receive a message

- ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio);

- ssize_t mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio, const struct timespec *abs_timeout);

More functions for message queues

- int mq_close(mqd_t mqdes);
- int mq_unlink(const char *name);
- int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
- int mq_setattr(mqd_t mqdes, const struct mq_attr *newattr, struct mq_attr *oldattr);

Messages: server

```
#include "mq_common.h“
int main(int argc, char **argv)
{
   mqd_t mq;
   struct mq_attr attr;
   char buffer[MAX_SIZE + 1];
   int must_stop = 0;
   attr.mq_flags = 0;
   attr.mq_maxmsg = 10;
   attr.mq_msgsize = MAX_SIZE;
   attr.mq_curmsgs = 0;
   mq = mq_open(QUEUE_NAME, O_CREAT | O_RDONLY, 0644, &attr);
   CHECK((mqd_t)-1 != mq);
   do { ssize_t bytes_read;
      bytes_read = mq_receive(mq, buffer, MAX_SIZE, NULL);
      CHECK(bytes_read >= 0);
      buffer[bytes_read] = '\0';
      if (! strncmp(buffer, MSG_STOP, strlen(MSG_STOP)))  must_stop = 1;
        else  printf("Received: %s\n", buffer);
   } while (!must_stop);
   CHECK((mqd_t)-1 != mq_close(mq));
   CHECK((mqd_t)-1 != mq_unlink(QUEUE_NAME));
   return 0;
}
```

Messages: client

```
#include "mq_common.h"
int main(int argc, char **argv)
{
   mqd_t mq;
   char buffer[MAX_SIZE];
   mq = mq_open(QUEUE_NAME, O_WRONLY);
   CHECK((mqd_t)-1 != mq);
   printf("Send to server (enter \"exit\" to stop it):\n");
   do {
      printf("> ");
      fflush(stdout);
      memset(buffer, 0, MAX_SIZE);
      fgets(buffer, MAX_SIZE, stdin);
      CHECK(0 <= mq_send(mq, buffer, MAX_SIZE, 0));
   } while (strncmp(buffer, MSG_STOP, strlen(MSG_STOP)));
  CHECK((mqd_t)-1 != mq_close(mq));
   return 0;
}
```

Messages: common

```
#ifndef COMMON_H_
#define COMMON_H_

#define QUEUE_NAME  "/test_queue"
#define MAX_SIZE    1024
#define MSG_STOP    "exit"

#define CHECK(x) \
   do { \
      if (!(x)) { \
         fprintf(stderr, "%s:%d: ", __func__, __LINE__); \
         perror(#x); \
         exit(-1); \
      } \
   } while (0) \


#endif /* #ifndef COMMON_H_ */
```