POSIX IPC

- POSIX IPC Overview
- POSIX IPC Names
- Opening/Creating
- Deleting IPC Objects
- POSIX Shared Memory Overview
- Creating Shared Memory Objects
- Using Shared Memory Objects
- Removing Shared Memory Objects
- Choice of Shared Mapping Mechanism
- Semaphores Overview
- POSIX Semaphores Overview
- Semaphore API Overview
- Opening a Named Semaphore
- Discarding Semaphores
- Waiting on a Semaphore
- Posting a Semaphore
- Retrieving Semaphore Value
- Unnamed Semaphores
- Initializing a Unnamed Semaphore
- Destroying an Unnamed Semaphore
- Shared Memory Demo Log
- Shared Memory Demo Setup

- Shared Memory Demo Protocol
- Shared Memory Demo Header File
- Shared Memory Demo Server
- Shared Memory Demo Server Continued
- Shared Memory Demo Server Continued
- Shared Memory Demo Server Continued
- Shared Memory Demo Server Continued
- Shared Memory Demo Server Continued
- Shared Memory Demo Server Continued
- Shared Memory Demo Client
- Shared Memory Demo Client Continued
- Shared Memory Demo Client Continued
- Shared Memory Demo Client Continued
- POSIX Message Queues Overview
- Opening/Creating a POSIX Message Queue
- Closing a POSIX Message Queue
- Unlinking a POSIX Message Queue
- Message Queue Attributes
- Message Queue Attributes Continued
- Sending a Message Via a POSIX Message Queue
- Receiving a Message Via a POSIX Message Queue
- POSIX Message Queues Additional API

- Message Queue Demo Log
- Message Queue Demo Reader Program
- Message Queue Demo Reader Program Continued
- Message Queue Demo Writer Program
- Message Queue Demo Writer Program Continued
- References

POSIX IPC Overview

- Provides message queues, semaphores and shared memory.
- Similar but simpler than older System V IPC.
- Allows both named and unnamed objects.
- Permissions similar to file permissions (execute permissions inapplicable).
- Example using POSIX shm and semaphores with Unix ndbm to implement simple person-age database.
- POSIX message queues.
- POSIX message queue priority demo.

POSIX IPC Names

- Global names.
- A POSIX IPC name consists of a `/` followed by one or more non-slash characters. Examples: `/mymq`, `/prj4sem`, etc.
- Semantics of names other than above are undefined.
- Some systems allow path-like names with path-like semantics. Implies that standard names cannot be created by unprivileged programs. Hence isolate name-dependent code for maximal portability.

Opening/Creating

- `shm_open()`, `sem_open()`, `mq_open()` analogous to `open()` used for returning handle to new/existing IPC object.
- 2 arguments for an existing object, or 3 arguments for creating a new object.
- 2nd argument flags include `O_RDONLY`, `O_WRONLY`, `O_RDWR` or’d with flags like `O_CREAT`, `O_EXCL`.
- Corresponding close call (for mq’s and sem’s) or unmap call (for shm) frees up all resources associated within current process with IPC object.

Deleting IPC Objects

- For each POSIX IPC type, there is a unlink call which removed object once no processes are using it.
- IPC objects have kernel persistence: disappear after being unlinked or at system shutdown.

POSIX Shared Memory Overview

- Typical Unix system has 3 shared memory mechanisms: System V shared memory, `mmap()` based shared memory and POSIX shared memory (chronological order).

- Allows shared memory region for IPC between unrelated processes without needing an underlying file.

- In some implementations, shared memory regions mapped into filespace. For example, on Linux virtual `tmpfs` filesystem mounted at `/dev/shm`.

- Use of shared memory involves using `shm_open()` to get a file descriptor and then use `mmap()` on the file descriptor with the `MAP_SHARED` flag.

- Can use other file descriptor calls (like `fstat()` `and ftruncate()`‘) on shm file descriptor.

Creating Shared Memory Objects

```
int shm_open(const char *posixName, int flags, mode_t mode);
```

posixName

Specifies name for shm object. Should meet restrictions for a POSIX IPC object name.

flags

Specifies one of O_RDONLY or O_RDWR or’d with O_CREAT, O_EXCL or O_TRUNC. Because of the need for synchronization, and also because some MMUs don’t support write-only memory, O_WRONLY is not allowed.

mode

Specifies permissions for shm object when created (must be specified as 0 even when not being created). Actually permissions affected by process umask.

On creation, user/group ownership as per effective [ug]id.

Close on exec (FD_CLOEXEC) flag set so that object is automatically closed on an exec().

Using Shared Memory Objects

- Newly created shared memory objects have size 0.
- Newly created or existing shared memory objects can be resized using the `ftruncate()` call.
- Newly added bytes are initialized to 0.
- Use `mmap()` call with file descriptor corresponding to POSIX shm object and flag `MAP_SHARED`.
- When done, unmap from process address space using `munmap()`.

Removing Shared Memory Objects

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

- Once object is unlinked, no further attempts to `shm_open()` without `O_CREAT` will succeed.
- Actual object is removed only after all processes have unmapped it.

Choice of Shared Mapping Mechanism

- System V shared memory readily available but not file descriptor based. Not possible resize shared memory.

- Shared file mapping provides persistent mapping (contents survive system restarts).

- POSIX shared memory provides only kernel persistence.

Semaphores Overview

- A semaphore has a value not allowed to fall below zero.
- Test whether decrement is permissible and actual decrement is **atomic**.
- An attempt to lower the semaphore value below 0 will block or return an error.
- Semaphore value can indicate number of available units of some resource.
- Semaphore decrement operation referred to as `P()` and increment operation referred to as `V()`.
- If semaphore value restricted to 0 or 1, then we have a *binary semaphore* as opposed to the more general *counting semaphore*.
- Used for synchronization.

POSIX Semaphores Overview

- Both named and unnamed counting semaphores.
- Named semaphores can be used between unrelated processes.
- Unnamed semaphores can be used for synchronization between the threads of a single process (in which case they can reside anywhere within the address space of that process) or for synchronization between multiple processes (in which case they must reside within shared memory (POSIX, System V, or mmap)).
- Semaphores are referred to using opaque `sem_t *` handles.

Semaphore API Overview

`sem_open()`

Returns handle to initialized named semaphore.

`sem_init()`

Initializes an unnamed semaphore.

`sem_post()`, `sem_wait()`

Resp. increments/decrements semaphore.

`sem_getvalue()`

Returns current semaphore value.

`sem_unlink()`

Removes a named semaphore.

`sem_destroy()`

Removes a unnamed semaphore.

Opening a Named Semaphore

```
sem_t *sem_open(const char *posixName, int flags, ...
                /* mode_t mode, unsigned int value */ );
```

- Arguments as expected.
- Last two optional arguments required when creating a semaphore.
- Returns `SEM_FAILED` (`((sem_t *)0)` or `((sem_t *)1)` (Linux)) on error.
- Cannot operate on copy of storage pointed to by returned value.
- Semaphores inherited across `fork()`. Can be used to synchronize parent and child.

Discarding Semaphores

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

Removes association between process and semaphore.

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

Removes mapping from `posixName` to semaphore. Removes semaphore if no processes have it open.

Waiting on 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);
```

- Decrements semaphore value by 1.
- `sem_wait()` blocks until decrement possible; `sem_trywait()` is a non-blocking version; `sem_timedwait()` blocks until decrement possible or timeout.
- Timeout argument `abs_timeout` to `sem_timedwait()` specifies timeout in absolute seconds and nanoseconds since the Epoch.
- Returns 0 on success, -1 on error. For `sem_trywait()`, sets `errno` to `EAGAIN` if decrement cannot be performed immediately. For `sem_timedwait()`, sets `errno` to `ETIMEDOUT` if timeout expires.
- If interrupted by a signal handler when blocked, then fails with error `EINTR`.

Posting a Semaphore

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

- Increments semaphore value by 1.
- If semaphore value was 0 before call, then some process blocked on the semaphore will be woken up. If multiple processes, then choice of process is indeterminate unless default round-robin scheduling; under real-time scheduling highest priority process which has been waiting longest is chosen.

Retrieving Semaphore Value

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

- Returns semaphore value in `*sval`.
- Returned semaphore value may be not current by the time the call returns.
- If semaphore value is 0, then returned semaphore value is 0 or negative of number of processes blocked on semaphore.

Unnamed Semaphores

- A semaphore shared among multiple threads can be allocated as a global variable or on the heap without giving it a name.
- A semaphore shared between related processes can be shared without needing a name when allocated in shared memory.
- When semaphores are used for protecting access to portions of a dynamic data structure liked a linked list, convenient to not have a name.

Initializing a Unnamed Semaphore

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

- `pshared` is non-zero, then shared among multiple processes; if 0, then shared only among threads of a single process.

- Permissions of unnamed semaphore are permissions of underlying memory region.

- Operations should not be performed on copy of initialized semaphore.

- Return 0 on success, -1 on error.

Destroying an Unnamed Semaphore

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

- `*sem` must refer to a previously initialized unnamed semaphore initialized using `sem_init()`.
- Once destroyed, the semaphore can be re-initialized using `sem_init()`.
- As usual, it would be an error to destroy if `sem` is a dangling pointer.
- Omitting `sem_destroy()` can result in resource leaks.

Shared Memory Demo Log

```
$ ./shmserve &
[1] 26813
$ memory attached at 0x7fa431a91000

$ ./shmclient
memory attached at 0x7f7c1ba3f000
?jim
not found
+jim 22
ok
?jim
22
+bill 11
ok
?bill
11
-jim
ok
?jim
not found
+bill 33
ok
```

```
?bill
33
^D
$
```

Shared Memory Demo Setup

- Simulate iterative server.
- Server creates shared memory segment + `SERVER`, `REQUEST` and `RESPONSE` semaphores on startup. `SERVER` semaphore initialized to 1, `REQUEST` and `RESPONSE` semaphores to 0.
- Uses Unix key-value db ndbm (other versions dbm, gdbm). Key routines `dbm_open()`, `dbm_fetch()`, `dbm_store()`, `dbm_delete()`.

Shared Memory Demo Protocol

1. Server waits on `REQUEST` sem

2. After client reads a line from stdin, it waits on `SERVER` semaphore. When the wait returns, other clients are locked out.

3. Client transfers request to shm and posts `REQUEST` semaphore. Waits on `RESPONSE` semaphore.

4. Server wakes up from `REQUEST` wait. Handles request and writes response to shm. Posts `RESPONSE` semaphore and loops to 1.

5. Client wakes up from `RESPONSE` wait. Copies result from shm to standard output. Posts `SERVER` semaphore and loops to 2.

Shared Memory Demo Header File

./programs/shmdemo/shmdemo.h

```
#define MAX_BUF 128

#define POSIX_IPC_NAME_PREFIX "/umrigar-"

#define SHM_NAME POSIX_IPC_NAME_PREFIX "shm"
#define SERVER_SEM_NAME POSIX_IPC_NAME_PREFIX "server"
#define REQUEST_SEM_NAME POSIX_IPC_NAME_PREFIX "request"
#define RESPONSE_SEM_NAME POSIX_IPC_NAME_PREFIX "response"

enum {
  SERVER_SEM,
  REQUEST_SEM,
  RESPONSE_SEM,
  N_SEMS
};

#define ALL_RW_PERMS    (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWGRP)

//Debugging code not shown
```

Shared Memory Demo Server

./programs/shmdemo/shmserve.c

```
#define ERROR "error"
#define NOT_FOUND "not found"
#define OK "ok"

#define DBM_NAME "mydbm"
```

Shared Memory Demo Server Continued

```
typedef struct {
  const char *posixName;
  int oflags;
  mode_t mode;
  unsigned initValue;
} SemOpenArgs;

static SemOpenArgs semArgs[] = {
  { .posixName = SERVER_SEM_NAME,
    .oflags = O_RDWR|O_CREAT,
    .mode = ALL_RW_PERMS,
    .initValue = 1,
  },
  { .posixName = REQUEST_SEM_NAME,
    .oflags = O_RDWR|O_CREAT,
    .mode = ALL_RW_PERMS,
    .initValue = 0,
  },
  { .posixName = RESPONSE_SEM_NAME,
    .oflags = O_RDWR|O_CREAT,
    .mode = ALL_RW_PERMS,
    .initValue = 0,
  },
};
```

Shared Memory Demo Server Continued

```
int
main(int argc, const char *argv[])
{
  DBM *dbm = dbm_open(DBM_NAME, O_RDWR | O_CREAT, 0660);
  sem_t *sems[N_SEMS];
  if (!dbm) fatal("could not open %s:", DBM_NAME);
  for (int i = 0; i < N_SEMS; i++) {
    const SemOpenArgs *p = &semArgs[i];
    if ((sems[i] = sem_open(p->posixName, p->oflags, p->mode, p->initValue))
        == NULL) {
      fatal("cannot create semaphore %s:", p->posixName);
    }
  }
```

Shared Memory Demo Server Continued

```
int fd = shm_open(SHM_NAME, O_RDWR|O_CREAT, ALL_RW_PERMS);
if (fd < 0) fatal("cannot create shm %s:", SHM_NAME);
if (ftruncate(fd, MAX_BUF) < 0) {
  fatal("cannot size shm %s to %d:", SHM_NAME, MAX_BUF);
}
char *buf = NULL;
if ((buf = mmap(NULL, MAX_BUF, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0))
    == MAP_FAILED) {
  fatal("cannot mmap shm %s:" SHM_NAME);
}
fprintf(stderr, "memory attached at %p\n", buf);
```

Shared Memory Demo Server Continued

```
while (1) { //server loop
  datum resp = { OK, strlen(OK) };
  if (sem_wait(sems[REQUEST_SEM]) < 0) {
    fatal("wait error on sem %s:", REQUEST_SEM_NAME);
  }
  switch (buf[0]) {
    case ’?’: {
      datum key = { buf + 1, strlen(buf) - 1 };
      resp = dbm_fetch(dbm, key);
      if (resp.dptr == NULL) {
        resp.dptr = NOT_FOUND; resp.dsize = strlen(NOT_FOUND);
      }
    }
    break;
```

Shared Memory Demo Server Continued

```
  case ’+’: {
    char *const p = strchr(buf, ’ ’);
    datum key = { buf + 1, p - (buf + 1) };
    datum content = { p + 1, strlen(buf) - (p - buf) };
    if (dbm_store(dbm, key, content, DBM_REPLACE) < 0) {
      resp.dptr = ERROR; resp.dsize = strlen(ERROR);
    }
  }
  break;
  case ’-’: {
    datum key = { buf + 1, strlen(buf) - 1 };
    if (dbm_delete(dbm, key) < 0) {
      resp.dptr = ERROR; resp.dsize = strlen(ERROR);
    }
  }
  break;
  default:
    resp.dptr = ERROR; resp.dsize = strlen(ERROR);
} /* switch (buf[0]) */
```

Shared Memory Demo Server Continued

```
    sprintf(buf, "%.*s", resp.dsize, resp.dptr);
    if (sem_post(sems[RESPONSE_SEM]) < 0) {
      fatal("cannot post sem %s:", RESPONSE_SEM_NAME);
    }
  } // while (1) server loop
  return 0;
}
```

Shared Memory Demo Client

./programs/shmdemo/shmclient.c

```
typedef struct {
  const char *posixName;
  int oflags;
} SemOpenArgs;

static SemOpenArgs semArgs[] = {
  { .posixName = SERVER_SEM_NAME,
    .oflags = O_RDWR,
  },
  { .posixName = REQUEST_SEM_NAME,
    .oflags = O_RDWR,
  },
  { .posixName = RESPONSE_SEM_NAME,
    .oflags = O_RDWR,
  },
};
```

Shared Memory Demo Client Continued

```
static void
semWait(sem_t *sem, const char *posixName)
{
  if (sem_wait(sem) < 0) {
    fatal("cannot wait on sem %s:", posixName);
  }
}

static void
semPost(sem_t *sem, const char *posixName)
{
  if (sem_post(sem) < 0) {
    fatal("cannot post sem %s:", posixName);
  }
}
```

Shared Memory Demo Client Continued

```
int
main(void)
{
  sem_t *sems[N_SEMS];
  for (int i = 0; i < N_SEMS; i++) {
    const SemOpenArgs *p = &semArgs[i];
    if ((sems[i] = sem_open(p->posixName, p->oflags)) == NULL) {
      fatal("cannot open semaphore %s:", p->posixName);
    }
  }
  int fd = shm_open(SHM_NAME, O_RDWR, 0);
  if (fd < 0) fatal("cannot open shm %s:", SHM_NAME);
  char *buf;
  if ((buf = mmap(NULL, MAX_BUF, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0))
        == MAP_FAILED) {
    fatal("cannot mmap shm %s:" SHM_NAME);
  }
  fprintf(stderr, "memory attached at %p\n", buf);
```

Shared Memory Demo Client Continued

```
  char *line = NULL;
  int lineSize = 0;
  int n;
  while ((n = getLine(stdin, &line, &lineSize)) != 0) {
    if (line[n - 1] == ’\n’) line[n - 1] = ’\0’;
    semWait(sems[SERVER_SEM], SERVER_SEM_NAME);
    strcpy(buf, line);
    semPost(sems[REQUEST_SEM], REQUEST_SEM_NAME);
    semWait(sems[RESPONSE_SEM], RESPONSE_SEM_NAME);
    printf("%s\n", buf);
    semPost(sems[SERVER_SEM], SERVER_SEM_NAME);
  }
  free(line);
  return 0;
}
```

POSIX Message Queues Overview

- Preserves message boundaries.
- Can be used without needing external synchronization.
- Messages have associated priorities with higher priority messages being received before lower priority messages.
- API includes `mq_open()`, `mq_send()`, `mq_receive()`, `mq_close()`, `mq_unlink()`, `mq_[gs]et_attr()` and `mq_notify()`.

Opening/Creating a POSIX Message Queue

```
mqd_t mq_open(const char *posixName, int oflags, ...
              /* mode_t mode, struct mq_attr *attr */);
```

- Additional arguments required when `oflags & O_CREAT` non-zero.

- `oflags` can include `O_NONBLOCK` for a non-blocking queue.

- If `attr` NULL, then queue created with implementation defined default attributes. Otherwise, `mq_open()` can use `attr` to specify the maximum # of messages on the queue and the maximum size of each message (upto system limits).

- Returns `mqd_t` (which could be an integral type or a pointer type), `(mqd_t)-1` on error.

- Child inherits open message queues across `fork()`; `exec()` or program termination closes all message queues.

Closing a POSIX Message Queue

```
int mq_close(mqd_t mqdes);
```

- Decrements open count associated with message queue underlying `mqdes`.
- Returns 0 on success, -1 on error.
- Program termination or `exec()` causes automatic closing of all message queues.
- Queue still exists until explicitly unlinked.

Unlinking a POSIX Message Queue

```
int mq_unlink(const char *posixName);
```

- Removed `posixName` message queue from POSIX namespace.
- Actual message queue deleted only when its open count drops to 0.
- Returns 0 on success, -1 on error.

Message Queue Attributes

```
struct mq_attr {
  long mq_flags;        /* Message queue description flags: 0 or
                           O_NONBLOCK [mq_getattr(), mq_setattr()] */
  long mq_maxmsg;       /* Maximum number of messages on queue
                           [mq_open(), mq_getattr()] */
  long mq_msgsize;      /* Maximum message size (in bytes)
                           [mq_open(), mq_getattr()] */
  long mq_curmsgs;      /* Number of messages currently in queue
                           [mq_getattr()] */
};

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);
```

Message Queue Attributes Continued

- `mq_setattr()` allows retrieving old attributes if `oldattr` is not `NULL`.
- `mq_curmsgs` may no longer be current by the time `mq_getattr()` returns.
- Only flag for `mq_flags` is `O_NONBLOCK` (can be changed using `mq_setattr()`.

Sending a Message Via a POSIX Message Queue

```
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
            unsigned int msg_prio);
```

- Sends message `*msg_ptr` of size `msg_len` to message queue specified by `mqdes` with priority `msg_prio`.

- Message size `msg_len` must be less that maximum message size of queue (available as `mq_msgsize` attribute of POSIX message queue).

- Higher priority (larger numbers) messages delivered before lower priority (smaller numbers) messages. Within same priority level, messages delivered FIFO.

- If no priority needed, always specify `msg_prio` as 0.

- Blocks if queue full (too many messages or size of all messages greater than some max), unless `O_NONBLOCK` in which case `errno == EAGAIN`.

Receiving a Message Via a POSIX Message Queue

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

- Returns # of received bytes in message, -1 on error.
- Uses `*msg_prio` (if not NULL), to return priority of received message.
- Irrespective of size of received message, `msg_len` must not be greater than `mq_msgsize` attribute of queue.

POSIX Message Queues Additional API

- Supports send/receive with timeout using `mq_timedsend()` and `mq_timedreceive()` with additional `const struct timespec *abs_timeout` argument.

- Allows asynchronous notification when a message arrives in a empty queue either in a separate thread or in a signal handler.

Message Queue Demo Log

```
$ ./mqreader &
[3] 21974
umrigar@zdu-laptop:mqdemo$ ./mqwriter
2 Message 1
received message: priority 2; contents  Message 1

2 Message 1
5 Message 2
received message: priority 5; contents  Message 2

received message: priority 2; contents  Message 1

$
```

Message Queue Demo Reader Program

./programs/mqdemo/mqreader.c

```
#define ALL_RW_PERMS (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)

int
main(int argc, const char *argv[])
{
  struct mq_attr mqAttr = {
    .mq_flags = 0,
    .mq_maxmsg = 5,
    .mq_msgsize = MQ_SIZE,
  };
  mqd_t mq =
    mq_open(MQ_NAME, O_RDONLY|O_CREAT, ALL_RW_PERMS, &mqAttr);
  if (mq == (mqd_t)-1) {
    fprintf(stderr, "cannot create mq %s:", MQ_NAME);
    exit(1);
  }
  while (1) {
    sleep(10);  //wait for messages to accumulate
```

Message Queue Demo Reader Program Continued

```
    while (1) {
      struct mq_attr currentAttr;
      if (mq_getattr(mq, &currentAttr) < 0) {
        fprintf(stderr, "cannot read attributes for mq %s:",
                MQ_NAME);
        exit(1);
      }
      if (currentAttr.mq_curmsgs == 0) break;
      char msg[MQ_SIZE];
      unsigned priority;
      int n = mq_receive(mq, msg, MQ_SIZE, &priority);
      if (n < 0) {
        fprintf(stderr, "receive error on mq %s:", MQ_NAME);
        exit(1);
      }
      printf("received message: priority %d; contents %.*s\n",
             priority, n, msg);
    } //while(1)
  } //while (1)
}
```

Message Queue Demo Writer Program

./programs/mqdemo/mqwriter.c

```
int
main(int argc, const char *argv[])
{
  mqd_t mq = mq_open(MQ_NAME, O_WRONLY);
  char *line = NULL;
  size_t lineSize = 0;
  while (1) {
    int priority;
    scanf("%d", &priority);
    if (priority < 0) break;
    int n = getline(&line, &lineSize, stdin);
    if (n == 0) break;
    if (n > MQ_SIZE) {
      fprintf(stderr, "line size %d exceeds max message size %d",
              n, MQ_SIZE);
    }
```

Message Queue Demo Writer Program Continued

```
    if (mq_send(mq, line, n, priority) < 0) {
      fprintf(stderr, "error sending to mq %s:", MQ_NAME);
      exit(1);
    }
  } //while (1)
  if (mq_close(mq) < 0) {
    fprintf(stderr, "cannot close mq %s:", MQ_NAME);
    exit(1);
  }
  free(line);
  return 0;
}
```

References

Text: Ch 51 - 54.