

InterProcess Communication

Shared Memory

Ken Gottry

Jan-2002

IPCS

The InterProcess Communication System (IPCS) is comprised of three services:

- Shared memory
- Semaphores
- Message Queues

IPCS is a standard feature of most UNIX systems including Solaris

This presentation focuses on shared memory

Shared Memory

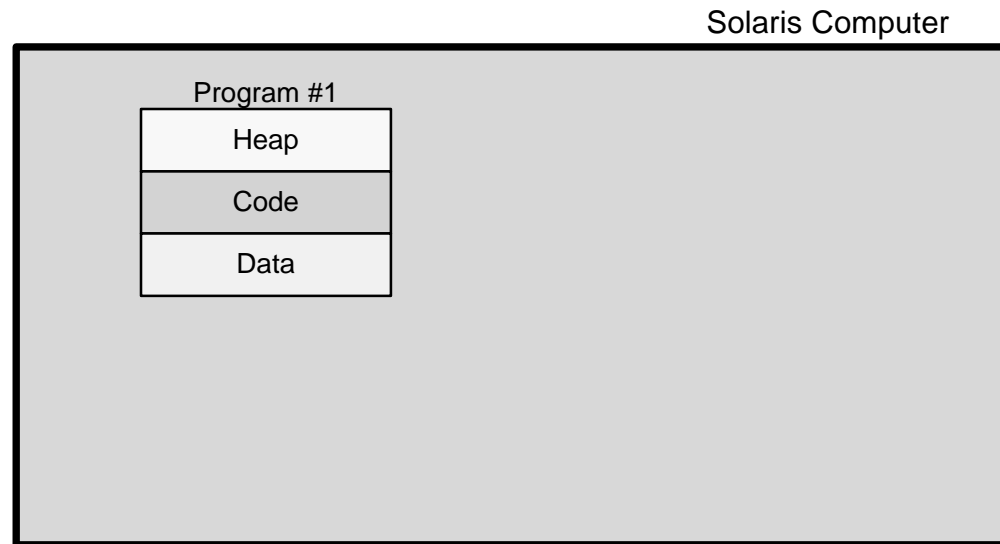
- Why you need it
- What it is
- How to use it
- Who uses it

Shared Memory - Why

Sections of a program

A program running on a Solaris computer consists of three sections:

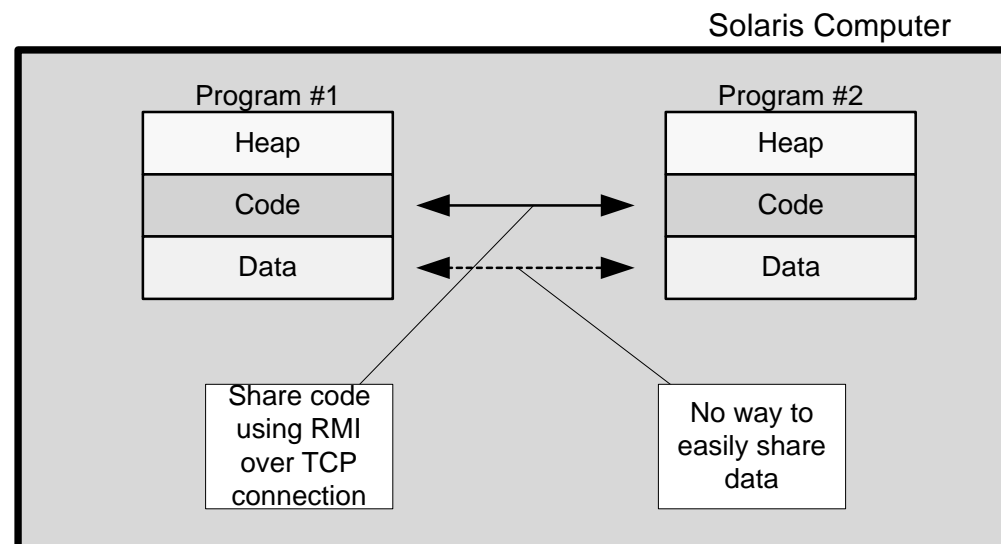
- Heap
- Code
- Data



Sharing code but not data

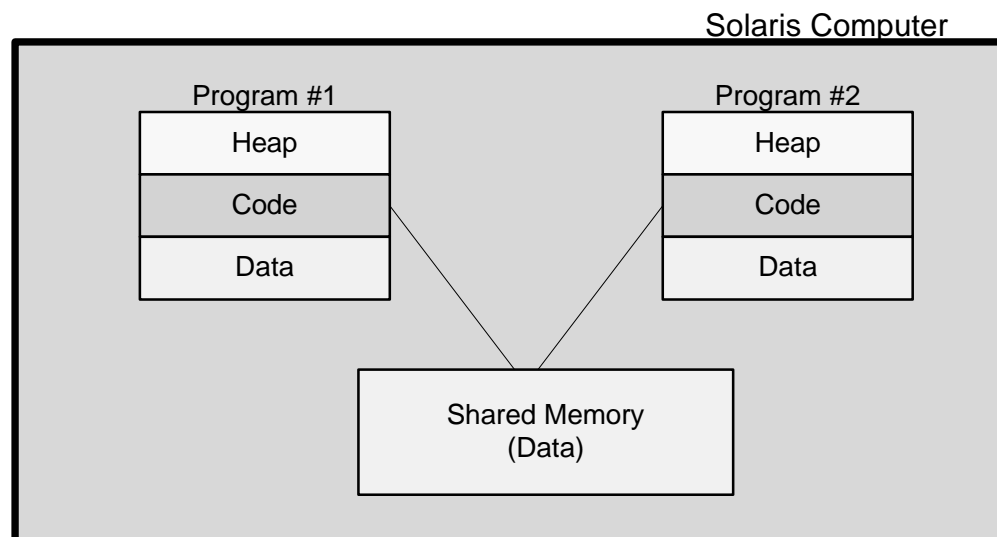
If a second program runs, it can share *code* with the first program through socket connections using things like RMI

However, there is no easy way to share *data* with the first program. That's where Shared Memory comes into play



Shared Memory - sharing data

- Multiple programs can read/write data from the shared memory area.
- Configuration parameters in `/etc/system` control size and use of shared memory
- Only works for programs running within the same computer



Shared Memory - What

What is shared memory

- Think of shared memory as an in-memory database
- Think of shared memory segments as database rows
- Shared memory segments have keys
- Shared memory segments contain data
- Parameters in `/etc/system` control shared memory
- One program can put (write) data into a shared memory segment and another program can get (read) data out of a shared memory segment
- When the program ends, the data in shared memory segments persists

- When the computer reboots, shared memory is erased and all data is lost!!

Can you see shared memory

- You can use the `ipcs -a` command to see the shared memory segments (see sample below)
- You must write a program to see the data within the shared memory segments

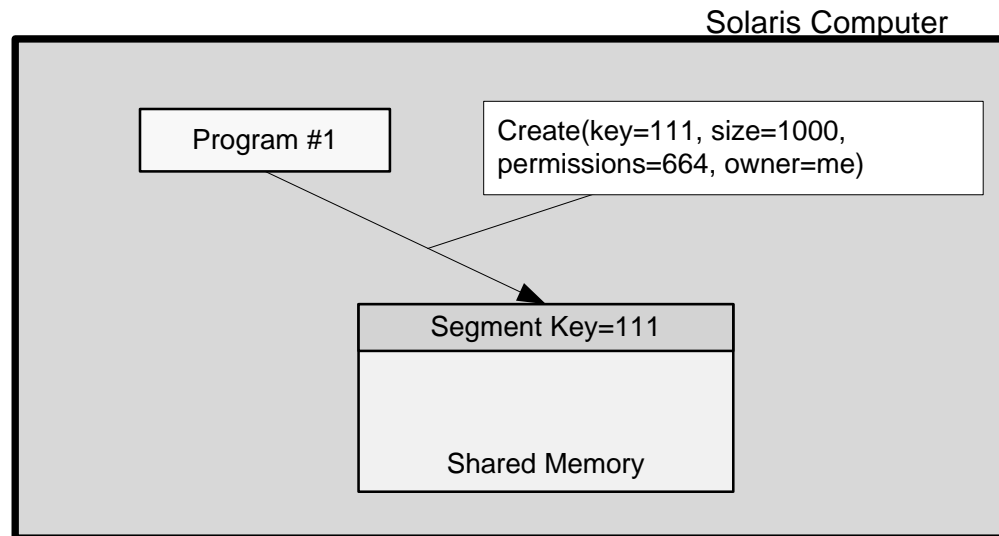
IPC status from <running system> as of Thu Oct 18 12:08:21 EDT 2001

T	ID	KEY	MODE	OWNER	GROUP	CREATOR	CGROUP	NATTCH	SEGSZ	CPID	LPID	ATIME
Shared Memory:												
m	0	0x400d7ca0	--rw-rw-rw-	root	other	root	other	15	1000	513	22347	11:42:20
m	9221	0xca892	--rw-rw-rw-	mqm	mqm	mqm	mqm	6	163824	12535	22347	11:42:20
m	9222	0xca8b0	--rw-rw-rw-	mqm	mqm	mqm	mqm	6	385768	12535	22347	11:42:20
m	9223	0xca8b2	--rw-rw-rw-	mqm	mqm	mqm	mqm	6	384284	12535	22347	11:42:20
m	9224	0xca8b3	--rw-rw-rw-	mqm	mqm	mqm	mqm	1	1636	12535	12541	16:30:10

Shared Memory - How

Creating a shared memory segment

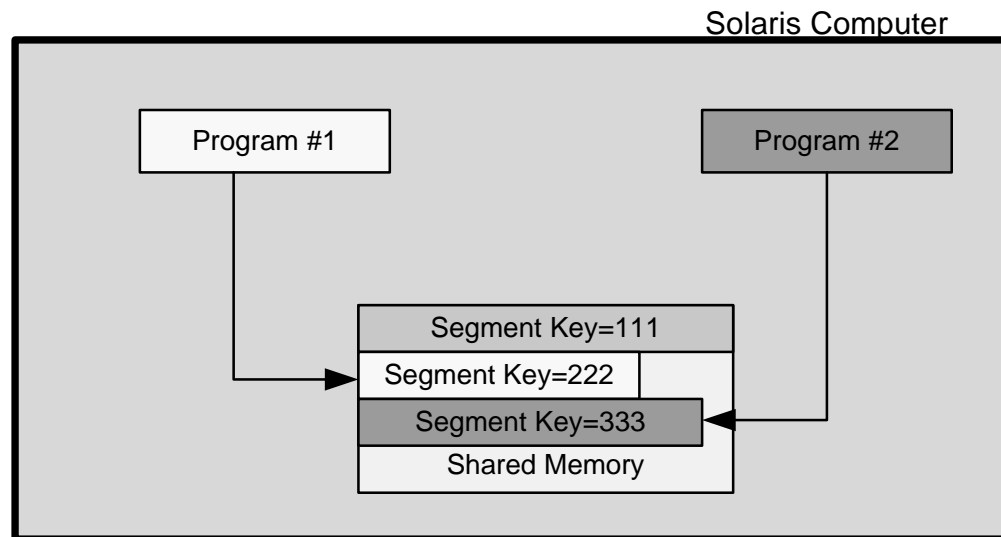
- Program #1 issues `shmget(..., ..., ...)` to create a shared memory segment with key of 111
- The ownership of the shared memory segment is the same as the ownership of Program #1
- The permissions (`rw-rw-r--`) are defined during the create



Note: the syntax used in these examples is not accurate. It is only used to show the type of parameters needed to perform the function

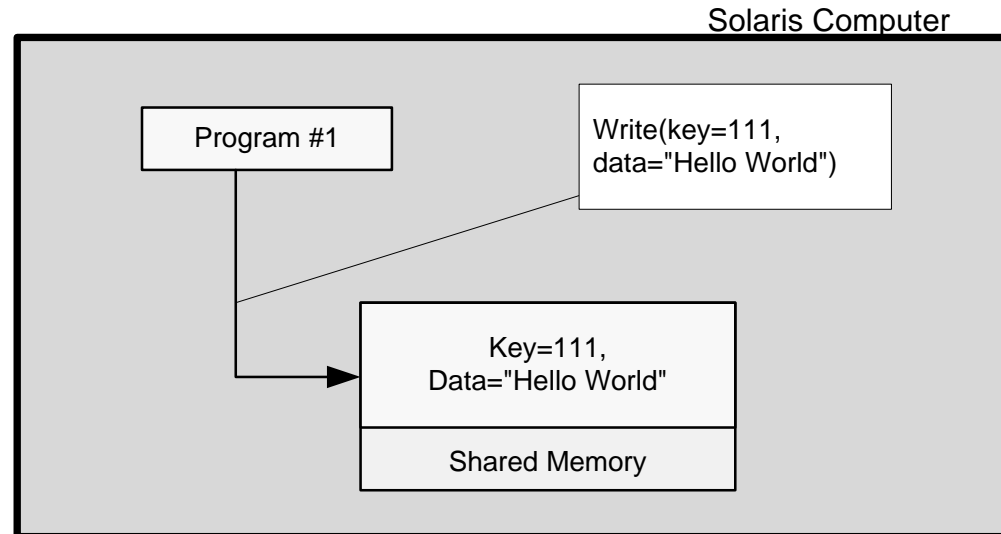
Many shared memory segments

- Shared memory can hold many segments created by many programs. Maximum = `shmsys:shminfo_shmmni`
- One process can create many segments. Maximum = `shmsys:shminfo_shmseg`
- Segment length can vary. Maximum = `shmsys:shminfo_shmmax`
- Run `ipcs -a` to see list of segments



Writing to a shared memory segment

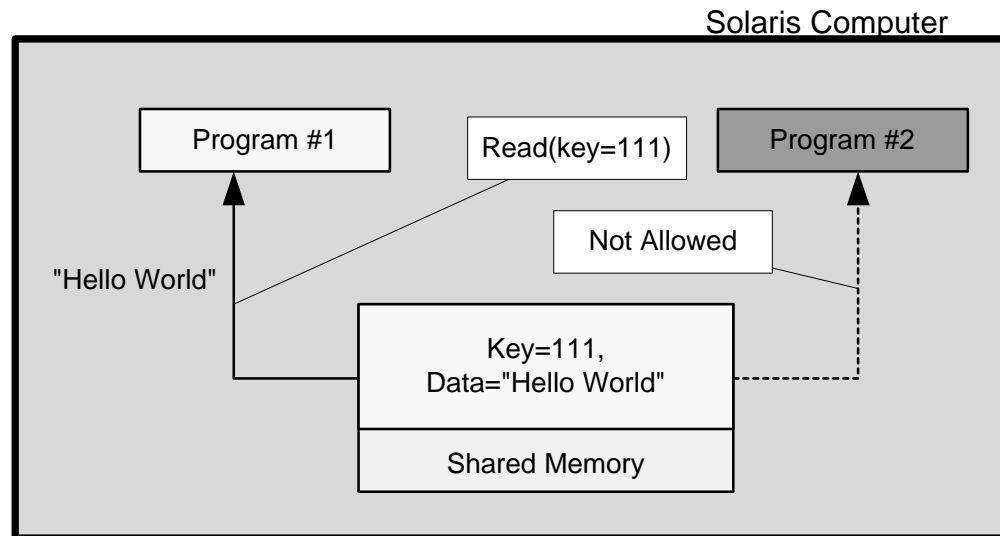
- A program can write data to a shared memory segment if it knows the key and if it has write permissions
- A program must first Attach to the shared memory segment. This is similar to Connect'ing to a database before writing a row



Note: the syntax used in these examples is not accurate. It is only used to show the type of parameters needed to perform the function

Reading data from shared memory

- A program can read data from a shared memory segment if it knows the key and if it has read permissions
- A program must first Attach to the shared memory segment. This is similar to Connect'ing to a database before reading a row



Note: the syntax used in these examples is not accurate. It is only used to show the type of parameters needed to perform the function

Synchronizing access to shared memory

- Since many programs can be reading and writing the same shared memory segment, there must be ways to synchronize access to the data
- There is a `shm_lock` and an `shm_unlock` command
- Similar concept to accessing a database

Shared Memory - Who

Examples of users of shared memory

- Oracle RDBMS
- MQSeries queue managers
- getAccess single signon software
- You - if you have a need and know how to

MQSeries uses shared memory

- Sample `ipcs -a` showing MQSeries queue managers
- Lots of shared memory segments of different sizes (SEGSZ)
- Lots of processes attached to each segment (NATTCH)
- Two semaphores (a different IPCS service)

```

IPC status from <running system> as of Tue Jan 1 16:08:24 EST 2002
T      ID      KEY      MODE      OWNER      GROUP  CREATOR  CGROUP  CBYTES  QNUM  QBYTES  LSPID  LRPID  STIME  RTIME  CTIME
Message Queues:
T      ID      KEY      MODE      OWNER      GROUP  CREATOR  CGROUP  NATTCH  SEGSZ  CPID  LPID  ATIME  DTIME  CTIME
Shared Memory:
m      0      0x400d7ca0 --rw-rw-rw-  mqm      mqm      mqm      mqm      13      1000   234 17783 16:04:55 16:07:23 12:37:38
m      1029   0xca892  --rw-rw-rw-  mqm      mqm      mqm      mqm      10      163824 3681 17783 16:04:55 16:07:23 15:54:32
m      1030   0xca8b0  --rw-rw-rw-  mqm      mqm      mqm      mqm      10      385768 3681 17783 16:04:55 16:07:23 15:54:32
m      1031   0xca8b2  --rw-rw-rw-  mqm      mqm      mqm      mqm      10      384284 3681 17783 16:04:55 16:07:23 15:54:32
m      1032   0xca8b3  --rw-rw-rw-  mqm      mqm      mqm      mqm      1      1636   3681 21914 15:54:32 19:34:15 15:54:32
m      1033   0xca8b7  --rw-rw-rw-  mqm      mqm      mqm      mqm      9      961584 3681 17783 16:04:55 16:07:23 15:54:32
m      1034   0xca8b9  --rw-rw-rw-  mqm      mqm      mqm      mqm      9      2001472 3681 17783 16:04:55 16:07:23 15:54:32
m      1035   0xca8be  --rw-rw-rw-  mqm      mqm      mqm      mqm      10     171852 3681 17783 16:04:55 16:07:23 15:54:32
m      1036   0xca8c2  --rw-rw-rw-  mqm      mqm      mqm      mqm      4      2900   3681 21914 19:34:15 19:35:56 15:54:32
m      1037   0xca8c5  --rw-rw-rw-  mqm      mqm      mqm      mqm      4      1416   3681 21914 19:34:15 19:35:56 15:54:32
m      1051   0xca8cc  --rw-rw-rw-  mqm      mqm      mqm      mqm      4      120384 3681 21914 19:34:15 19:35:56 15:54:33
m      1052   0xca8cd  --rw-rw-rw-  mqm      mqm      mqm      mqm      2      1050160 3681 3687 13:36:57 13:37:07 15:54:33
T      ID      KEY      MODE      OWNER      GROUP  CREATOR  CGROUP  NSEMS  OTIME  CTIME
Semaphores:
s      0      0xd7ca0  --ra-ra-ra-  mqm      mqm      mqm      mqm      1 16:04:55 12:37:38
s      3      0xd3e0d  --ra-ra-ra-  mqm      mqm      mqm      mqm      2 16:07:23 12:37:39

```

How MQSeries uses shared memory

- The top box shows `ipcs -a` output
- The bottom box shows `ps -ef` for MQSeries queue managers
- The Creator Process ID (**CPID**) tells you which process created the shared memory segment
- The Last Process ID (**LPID**) tells you which process last accessed the shared memory segment and when (**ATIME**)

```
IPC status from <running system> as of Tue Jan 1 16:33:26 EST 2002
T      ID      KEY      MODE      OWNER      NATTCH      SEGSZ      CPID      LPID      ATIME
Shared Memory:
m      0      0x400d7ca0 --rw-rw-rw-      mqm      13      1000      234 18042 16:32:21
m      1029    0xca892  --rw-rw-rw-      mqm      10      163824    3681 18042 16:32:21
m      1030    0xca8b0  --rw-rw-rw-      mqm      10      385768    3681 18042 16:32:21
m      1031    0xca8b2  --rw-rw-rw-      mqm      10      384284    3681 18042 16:32:21
m      1032    0xca8b3  --rw-rw-rw-      mqm      1      1636      3681 21914 15:54:32
```

```
mqm 18042 3686 0 16:36:15 ?      0:00 /opt/mqm/bin/runmqchl -c EQDEVAQ1.TO.QMBILL -m EQDEVAQ1
mqm 3685 3681 0 Dec 17 ?      0:00 /opt/mqm/bin/amqrrmfa -t2332800 -g5184000 -c3600 -m EQDEVAQ
mqm 3687 3681 0 Dec 17 ?      0:08 amqzlaa0 -mEQDEVAQ1 -fip0
mqm 3681 1 0 Dec 17 ?      0:02 amqzxma0 -m EQDEVAQ1
mqm 3684 3681 0 Dec 17 ?      0:02 amqzllp0 -mEQDEVAQ1 ?
```