ECE 3574: InterProcess Communication using Shared Memory

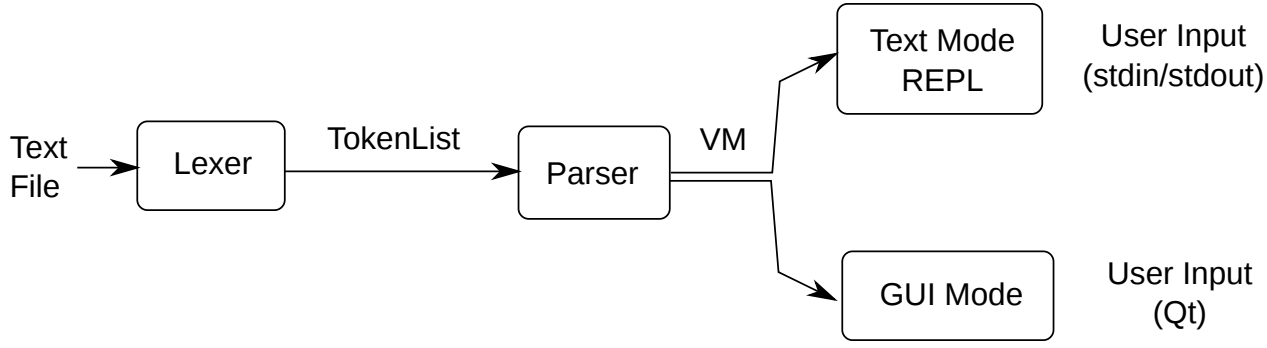
Changwoo Min

Project milestones

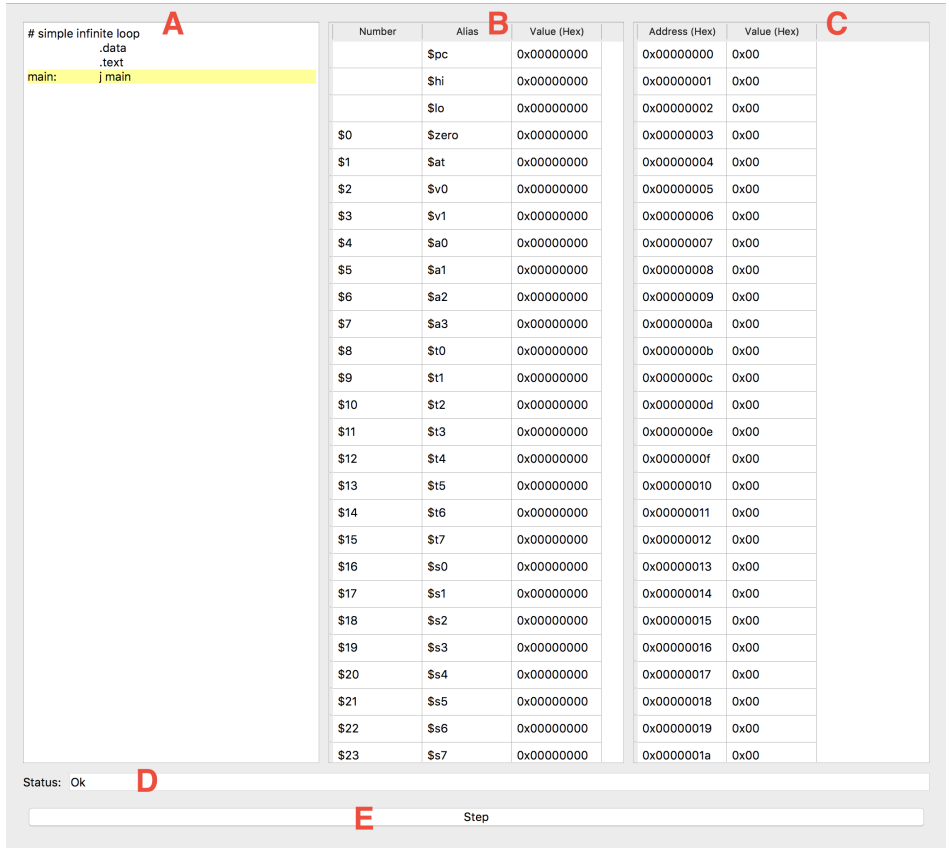
Milestone	Duration	Points
Milestone 0	3 weeks	20
Milestone 1	3 weeks	48
Milestone 2	4 weeks	70
Milestone 3	**2 weeks**	**92**
Milestone 4	?	?

Milestone 3

- **Add a Qt-based GUI to simmips**
  - Due: 4/9
  - Specification



SIMMIPS GUI


A
B
C
simple infinite loop
.data
.text
main: j main
Number
Alias
Value (Hex)
$pc 0x00000000
$hi 0x00000000
$lo 0x00000000
$0 $zero 0x00000000
$1 $at 0x00000000
$2 $v0 0x00000000
$3 $v1 0x00000000
$4 $a0 0x00000000
$5 $a1 0x00000000
$6 $a2 0x00000000
$7 $a3 0x00000000
$8 $t0 0x00000000
$9 $t1 0x00000000
$10 $t2 0x00000000
$11 $t3 0x00000000
$12 $t4 0x00000000
$13 $t5 0x00000000
$14 $t6 0x00000000
$15 $t7 0x00000000
$16 $s0 0x00000000
$17 $s1 0x00000000
$18 $s2 0x00000000
$19 $s3 0x00000000
$20 $s4 0x00000000
$21 $s5 0x00000000
$22 $s6 0x00000000
$23 $s7 0x00000000
Address (Hex)
Value (Hex)
0x00000000 0x00
0x00000001 0x00
0x00000002 0x00
0x00000003 0x00
0x00000004 0x00
0x00000005 0x00
0x00000006 0x00
0x00000007 0x00
0x00000008 0x00
0x00000009 0x00
0x0000000a 0x00
0x0000000b 0x00
0x0000000c 0x00
0x0000000d 0x00
0x0000000e 0x00
0x0000000f 0x00
0x00000010 0x00
0x00000011 0x00
0x00000012 0x00
0x00000013 0x00
0x00000014 0x00
0x00000015 0x00
0x00000016 0x00
0x00000017 0x00
0x00000018 0x00
0x00000019 0x00
0x0000001a 0x00
Status: Ok
D
E
Step

Why you should start early

- **Largest points for the shortest period of time**
  - We have only two weekends
- **Lots of integration challenges**
  - Refactoring VirtualMachine, and simmips
  - Debugging
  - Adding a Qt-based GUI
- **Start today**
  - Attend the TA recitation session today

How to debug in Linux: use `gdb`

- `gdb` : text mode debugger in Linux

```
$> mkdir build
$> cd build
$> cmake -DCMAKE_BUILD_TYPE=Debug ..  # debug build for debugging
$> make
$> gdb --args ./unit_tests "[parser]" # run gdb for a command
gdb> b parser.cpp:100                 # set a breakpoint at Line 100 in parser.cpp
gdb> run                              # actually run ./unit_tests
gdb> layout src                       # When the break point his,
                                      # press Ctrl-x Ctrl-a to see the source code
gdb> n                                # next
gdb> s                                # step into
gdb> p VAR                            # print VAR
gdb> p *ADDR                          # print the contents at ADDR
gdb> quit                             # quit
```

- See gdb cheatsheet and GDB TUI commands

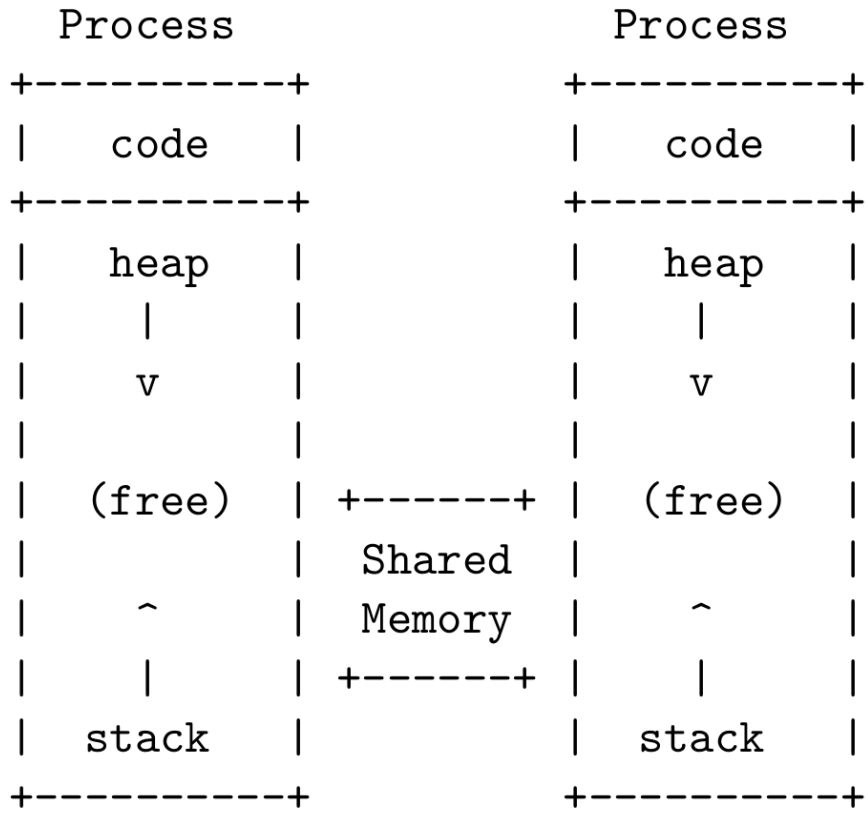
InterProcess Communication using Shared Memory

- Today we are going to see how processes can communicate using shared memory
  - Stack, Heap, and mapped memory segment
  - POSIX Shared Memory API
  - Windows Shared Memory API
  - Cross-platform shared memory using QSharedMemory
  - Boost interprocess library
  - Exercise

An alternative to IPC with messaging is to share memory

- Rather than send messages over pipes/sockets, we explicitly share memory.
- Fast, but requires explicit *synchronization*!
- This is the model that maps onto threads (which share a heap).
- Can still implement message passing on top of the shared memory

A revision of our process memory model



There are a few shared memory APIs on Unix

- We will look briefly at the POSIX shared memory API since it is the most portable.
- It is a generalization of memory mapped files.
- One process creates, the others attach. The kernel guarantees this operation is atomic.

POSIX shared memory API

- shm_open(): attach to an existing or create a new shared memory segment
- ftruncate(): size a shared segment
- mmap(): map a mapped object from caller's address space
- munmap(): unmap a mapped object from caller's address space
- close(): close file descriptor returned by shm_open()
- shm_unlink(): remove SHM object name, mark for deletion
- fstat(): retrieve stat structure describing objects

shm_open – open a shared memory object

```c
#include <sys/mman.h>
#include <fcntl.h>

int shm_open(const char *name, int oflag, ...);

/* Each shared segment has a name (called the key)
 * The oflag argument is an or'd combintation of
 *   O_RDONLY: open for reading only
 *   O_RDWR:   open for reading and writing
 *   O_CREAT:  create object if it does not exist
 *   O_EXCL:   error if create and object exists
 */
```

- Manual

ftruncate – truncate or extend a file to a specified length

- Here the file is a shared memory segment

```c
#include <unistd.h>
int ftruncate(int fildes, off_t length);

/* The first argument of the file descriptor returned from
 * shm_create().
 * The second argument is the new length in bytes.
 */
```

- Manual

mmap – allocate memory, or map files or devices into memory

```
#include <sys/mman.h>

void * mmap(void *addr, size_t len, int prot,
            int flags, int fd, off_t offset);

/* This call is used next to map the memory into
 * the address space of the process.
 *
 * Returns a pointer to the beginning of
 * the raw memory segment, the base pointer.
 */
```

- Manual

Once the mapping is made you can read (and if setup) write to the shared memory

- This required manual placement of objects in memory, offset from the base pointer.
- This can be tricky as it requires manually computing pointer offsets based on object size.
- We will see how to do this with “placement” new.
- Note, you are generally limited to plain-old-data (POD) types unless the objects are allocation aware. For STL containers you can write a custom allocator

When you are done you unmap the segment

```
#include <sys/mman.h>
int munmap(void *addr, size_t len);

/* After this access to the shared memory is an access violation
 * and will generate a seg fault. */
```

- Manual

Finally close it, using the file descriptor

```c
#include <unistd.h>

int close(int fildes);
```

- Manual

Lets look at an example

- See `posix_example/count.cpp`
  - `g++ -lrt count.cpp`

The Windows shared memory API is similar

- CreateFileMapping/OpenFileMapping replace shm_create
- MapViewOfFile replaces mmap
- UnmapViewOfFile replaces munmap
- CloseHandle replaces close

Qt provides a cross-platform abstraction QSharedMemory

- Shared memory with a locking mechanism.
- This makes synchronization easier. We will see how to do that ourselves next week.
- See example `qt_shared_deque`
  - QSharedMemory: Introduction, API, Example
  - std::is_pod

Another popular cross-platform IPC library is boost::interprocess

- It provides wrappers for the platform-specific shared memory APIs similar to QSharedMemory.
- It provides shared memory aware containers like vector.
- It also provides a key based object store in the shared memory segment.
- The key-based store provides the ability to easily create objects in a shared memory segment, giving a string name to them so that any other process can find, use and delete them from the segment when the objects are not needed anymore.

Exercise 19

- See website

Next Actions and Reminders

- Read about Message Serialization
- Milestone 3 released, due 4/9 by 11:59 pm.

ECE 3574: InterProcess Communication using Shared Memory

Changwoo Min