# Project 3: s2dsm (Super Simple Distributed Shared Memory) Version 2

- **Handed out:** Tue Feb 23, 2021
- **Modified:** Sun Mar 7, 2021
- **Due dates**
  - Part 1: Friday, March 05, 2021
  - Part 2: Friday, March 12, 2021
  - Part 3: Friday, March 19, 2021

## Introduction

The goal of this project is to implement `s2dsm`, a distributed shared memory (user-level) program that supports a page-granule MSI protocol using `userfaultfd`. Two instances of `s2dsm` will communicate their page information using sockets. Though `s2dsm` may run on different machines to support a (truely) distributed shared memory, in this project, we will run two `s2dsm` processes on a single machine for simplicity. This project consists of three parts.

## Recommended Background Reading

- Socket Programming in C/C++
- POSIX thread (pthread) libraries
- userfaultfd
- MSI protocol

## === Part 1 ===

### P1.1: Understanding userfaultfd demo source code and output

[10 points] The attached source code `uffd.tar.gz` is a simple application that demonstrates how `userfaultfd` can be used to handle page faults at the user level. It contains two files shown below.

```
$  tar xzvf uffd.tar.gz uffd/
uffd/
uffd/Makefile
uffd/uffd.c
```

Carefully read, compile and execute uffd.c file to understand how userfaultfd can be used to manage paging. Once you completely understand uffd.c, add comments of each line (H1-H10, M1-M7, U1-U7) explaining what the line means. For U1-U7, explain the output of `./uffd 1`.

If you see the error message "userfaultfd Function not implemented", then you should recompile your kernel with `CONFIG_USERFAULTFD=y`.

Create a gzip-ed tarball named `uffd.tar.gz`.

```
$  tar czvf uffd.tar.gz uffd/
uffd/
```

```
uffd/Makefile
uffd/uffd.c
```

## P1.2: Pairing memorey regions between two systems

[10 points] Create a (user-land) application named `s2dsm` that takes as input a local port number and a remote port number. Two instances of `s2dsm` communicates each other over sockets. For example, when running `./s2dsm 5000 6000` and `./s2dsm 6000 5000`, two processes pair up each other. The first `s2dsm` process listens on the port 5000 and send messages to the port 6000. The second process listens on the port 6000 and send messages to the port 5000.

After pairing, the first `s2dsm` process:

- asks a user to specify the number of pages to allocate through stdin: e.g., " > How many pages would you like to allocate (greater than 0)?"
- mmaps a memory region of the size specified by `<the number of pages> * PAGESIZE`;
- printfs the address of the mmaped region (the return value of mmap) and the mmapped size; and
- sends a message including the mmaped address and size to the second process over a socket communication.

The second `s2dsm` process:

- receives the message including the mmapped address and size from the first process;
- mmaps a memory region of the same size to the same address (by specifying the first argument of mmap); and
- printfs the address of the mmaped region and the mmapped size.

Program your code in `s2dsm.c` and create `Makefile`.

## P1.3: Test your code so far

[10 points] Once you get this far, take a screenshot of the printfs for memory address and size for mmaped region of two paired `s2dsm` processes. To do this, you must use tmux to split the terminal vertically, left-side running the first process, followed by the right side running the second process. The first process will ask the user to input a number of pages it should mmap. It will then mmap first, and communicate the mmap information to the second application. You must show to printf of mmap information from both applications. Name your screenshot as `s2dsm.png`.

## Submission

Make a folder named with your SBU ID (e.g., 112233445), put `uffd.tar.gz`, `Makefile`, `s2dsm.c`, and screenshot `s2dsm.png` files in the folder, create a single gzip-ed tarball named `[SBU ID].tar.gz`, and turn the gzip-ed tarball to Blackboard.

```
$  tar czvf 112233445.tar.gz 112233445/
112233445/
112233445/Makefile
112233445/s2dsm.c
112233445/s2dsm.png
112233445/uffd.tar.gz
```

## === Part 2 ===

### P2.1: Implement `userfaultfd`

[20 points] Picking up from Part 1, once you have a mmapped memory region, register the memory region with `userfaultfd`. Once paired, `s2dsm` should repeatly ask a user for two inputs:

- " > Which command should I run? (r:read, w:write): "
- " > For which page? (0-%i, or -1 for all): "

You will have to replace "%i" with the the number of pages specified by the user from Part 1.

The r (read) command reads the contents in t he specified (or all) pages. When a page fault occurs, printf `" [x] PAGEFAULT\n"`. For each page, printf the contents as follows: `" [*] Page %i:\n%s\n"` where `%i` is the page number and `%s` is the page content.

The w (write) command asks a user for the content to write on the specified (or all) pages:

- " > Type your new message: "

Write the new message into the specified (or all) page. Similar to the r command, when a page fault occurs, printf: `" [x] PAGEFAULT\n"`. Printf the (updated) contents as follows: `" [*] Page %i:\n%s\n"` where `%i` is the page number and `%s` is the page content.

Program your code in `s2dsm.c` and create `Makefile`.

### P2.2: Test your code so far

[10 points] Take a screenshot of the outputs (printfs) when following the operation sequence below:

- Pair the two application with 3 pages [0-2]
- Application 1 reads page 0
- Application 1 reads all pages
- Application 1 writes "Hello Userfaultfd!" to page 2
- Application 1 reads all pages
- Application 1 writes "Writing this to all pages!" to all pages
- Application 1 reads all pages

You only need to show the outputs of one process. Name your screenshot as `s2dsm.png`.

### Submission

Make a folder named with your SBU ID (e.g., 112233445), put `Makefile`, `s2dsm.c`, and screenshot `s2dsm.png` files in the folder, create a single gzip-ed tarball named `[SBU ID].tar.gz`, and turn the gzip-ed tarball to Blackboard.

```
$  tar czvf 112233445.tar.gz 112233445/
112233445/
112233445/Makefile
112233445/s2dsm.c
112233445/s2dsm.png
```

## === Part 3 ===

### P3.1: Implement MSI page coherence protocol

[80 points] Picking up from Part 2, you will implement the MSI protocol to ensure the coherence of a shared memory region between two processes. To do this, each process maintains an array, called `msi_array` that keeps track of MSI information (enum values: M, S, and I) for each page.

Once paired, `s2dsm` should repeatly ask a user for two inputs:

- " > Which command should I run? (r:read, w:write, v:view msi array): "
- " > For which page? (0-%i, or -1 for all): "

You will have to replace "%i" with the the number of pages specified by the user from Part 1. The new "v" command will allow the user to view the contents of the MSI array.

For the "r"ead and "w"rite commands, implement the MSI protocol as specified in MSI protocol Wikipedia Initially, all pages are in the invalid state (I).

`Notes (added on March 7)`:

- **Initial read protocol (Refer to the discussion @93 in Piazza):**
    - Given two processes A and B and page number 1:
    - Initially, A1 (A's page 1) and B1 (B's page 1) are INVALID.
    - Read (A1) will trigger a page fault (as this is the first access).
    - A detects that A1 is in the INVALID state.
    - A attempts to fetch B1 from B.
    - B informs A that B1 is also in the INVALID state (if B1 is not INVALID, then B simply sends back the page and updates the state accordingly).
    - When A finds that B1 is also INVALID, then it printf null string: e.g., printf("[*] Page %d:n%sn",1,"");
- **Invaidation protocol (@91 in Piazza):**
    - Given two processes A and B and page number 1:
    - When B receives the INVALIDATE request for page 1 from A (as A writes on A1), B should make B1 INVALID and call `madvise(B1, PAGE_SIZE, MADV_DONTNEED)`.
    - This will make any following access to B1 a page fault.
    - If B1 is in the INVALID state, B should start handling Read (B1) or Write (B1) from the page fault: e.g., Read(B1) attempts to fetch A1.
    - This will allow us to have an unified read fetch logic for an initial read (@93) and a read from a invalidated page (@91).
- **Concurrency protocol (@89 in Piazza):**
    - Assume there are no concurrent operations: B takes some actions after A completes its operation (and MSI protocol handling).

### P3.2: Test your code

[10 points] Run two applications as follows and take a screenshot of the outputs (printfs) from both applications. You may take multiple screenshots.

- Pair the two application with 3 pages [0-2]

- Application 1 views all MSI contents
- Application 2 views all MSI contents
- Application 1 reads page 0
- Application 1 views all MSI contents
- Application 2 views all MSI contents
- Application 1 reads all pages
- Application 1 views all MSI contents
- Application 2 views all MSI contents
- Application 2 writes "Hello Userfaultfd!" to page 2
- Application 1 views all MSI contents
- Application 2 views all MSI contents
- Application 1 wites "Writing this to all pages!" to all pages
- Application 1 views all MSI contents
- Application 2 views all MSI contents
- Application 1 reads all pages
- Application 1 views all MSI contents
- Application 2 views all MSI contents

## Submission

Make a folder named with your SBU ID (e.g., 112233445), put `Makefile`, `s2dsm.c`, and screenshot `s2dsm.png` files in the folder, create a single gzip-ed tarball named `[SBU ID].tar.gz`, and turn the gzip-ed tarball to Blackboard.

```
$  tar czvf 112233445.tar.gz 112233445/
112233445/
112233445/Makefile
112233445/s2dsm.c
112233445/s2dsm.png
```