Assignment 5 - Syscall Implementation

Vishal Vijay Devadiga (CS21BTECH11061)

Understanding syscalls

Syscalls are functions designed to have user programs interact with the operating system. Functions such as creating a new child process, waiting (sleep) and even small functions such as uptime and date are syscalls implemented in the operating system.

In xv6

Each syscall has a specific number assigned to it(in `syscall.h`), which is used to reference to that syscall being called.

All syscalls are defined in `sysfile.c` and `sysproc.c`, with further definitions of some functions used by the syscalls in `file.c` and `proc.c` respectively.

System calls available to user programs are prototyped in `user.h` and `usys.S`.

The flow of how a user programs calls a syscall is as follows:

Note: I am using the pre-compiled code file names. Assembly files do the actual job.

- User program imports `user.h`
- The said program calls for that system call.
- Flow: `vector.pl` –> `trapasm.S` –> `trap.c` –> `syscall.c`
- `syscall` function in `syscall.c` executes with the system call number stored in `proc->tf->eax` where `proc` is the user process and `tf` is the trapframe and `eax` is a register used for arguments and return values.
- `syscall` checks whether the number of the system call called by that user process is valid or not.
- It then executes the system call if valid and obtains the return value(stored in `proc->tf->eax`).
- It then hands back the control to the user program with that return value.

Observations

PART A

In **part A**, I avoided the printing for the system call `write`(syscall number 16) because it was being called a lot during printing to the terminal in other parts, due to which the output was not legible.

PART C

In **part C**, I have printed the the page entry number with its virtual address and physical address.

Physical address of the page is obtained by the page itself and its virtual address is obtained by the `pg_addr` function that uses its page directory index and page table index.

The `mypgtPrint.c` user program prints the user-accesible and valid page entries.

These results were observed when a array was declared by various means:

- No array is declared : 2 valid page entries(entry number 0 and 2).
- Local array is declared : 2 valid page entries(entry number 0 and 2).
- Global array is declared : 12 valid page entries(entry number 0-10 and 12).
- Array is declared by malloc : 12 valid page entries(entry number 0 and 2-12).

Virtual addresses of the page entries is from `0x0000` to `0xc000` and do not change, while physical addresses vary between each run of the command.

Reasoning of the experiments

A local array is declared in the stack, while the global array is declared in the data section and a malloced array is declared in the heap.

Stack size is fixed for each program. Stack frame size is determined at compile-time. The size depends on the size of the local variables. The memory for the array is allocated within the already allocated stack, thus, no additional page entries are needed.

Global arrays and malloced arrays require additional memory allocation and page table entries compared to local arrays because they are stored in the data segment and the heap segment of the program's memory space respectively.