Nome: Gustavo Bastos de Souza
Número USP: 14576642

2024/09/11

- Constante para *getreadcount()* em syscall.h
    - Número da syscall que indica posição no array de syscalls
- extern int sys_getreadcount(void) em syscall.c
    - Definição da função da syscall.
- [SYS_getreadcount] sys_getreadcount em syscall.c
    - Definir a syscall no array de syscalls
- Makefile modificado em UPROGS
    - Assim é possível chamar no qemu (*_getreadcount*\ indica que no qemu podemos chamar *getreadcount.c*)
- SYSCALL(getreadcount) em usys.S
    - Dessa forma a aplicação entende getreadcount como uma chamada de sistema e move valores nos registradores para executar trap

2024/09/13

- Pesquisa no repositório oficial do xv6 (https://github.com/mit-pdos/xv6-public) da chamada *getpid*, para assim identificar melhor o padrão das chamadas de sistema
- *int getreadcount(void)* em *user.h*
    - Assim foi entendido que à partir de sua definição o usuário pode acessar essas chamadas de sistemas (como utilizadas em *usertests.c* e nas próprias implementações que no Makefile permitem que o qemu chame as syscalls, como *getreadcount.c*)
- *int sys_getreadcount()* em *sysproc.c*. A função somente retorna a contagem da quantidade de *read*'s global (*globalReadCount*).

2024/09/14

- Definição de um *extern uint globalReadCount* em defs.h, será usado para soma a cada read em *read()* em *sys_read()* em *sysfile.c*
- Definição de um *extern struct spinlock readCountLock* para *acquire()* e *release()* quando for executado a contagem em *sys_read()* em *sysfile.c.*
    - Lock e suas funções usadas para evitar concorrência.
- Funcionando para os dois casos de teste!

2024/09/18

- Uso de *initlock()* graças ao email enviado pelo professor, explicando da necessidade de inicializar o lock antes de usá-lo em *sysfile.c* (dentro da função *sys_read()*).