CS 3214
Computer Systems

Virtual Memory - mmap

Godmar Back

Using mmap()

- mmap() is the Unix API by which a process can create mappings in its address space
  – Very powerful & flexible

PROT_READ
PROT_WRITE
PROT_EXEC

MAP_SHARED
MAP_PRIVATE

MAP_ANONYMOUS

MAP_FIXED

```
void *mmap(void *start, size_t length, int prot, int flags,
           int fd, off_t offset);

int munmap(void *start, size_t length);
```

mmap() for file I/O

```
int
main(int ac, char *av[])
{
    int fd = open(av[1], O_RDONLY);
    assert (fd != -1);

    off_t filesize = lseek(fd, 0, SEEK_END);
    assert (filesize != (off_t) -1);

    size_t pgsize = getpagesize();
    size_t mapsize = (filesize + pgsize - 1) & ~(pgsize-1);

    void *addr = mmap(NULL, mapsize, PROT_READ, MAP_PRIVATE, fd, 0);
    if (addr == MAP_FAILED) {
        perror("mmap"); exit(-1);
    }
    assert (close(fd) == 0);

    // access file data like memory
    char *start = addr, *end = addr + filesize;
    while (start < end)
        fputc(*start++, stdout);
    return 0;
}
```

mmap() for parent/child communication

```
int
main(int ac, char *av[])
{
    size_t sz = getpagesize();
    int sharedflag = ac < 2 || strcmp(av[1], "-private")
                        ?  MAP_SHARED : MAP_PRIVATE;
    void *addr = mmap(NULL, sz,
                      PROT_READ|PROT_WRITE,
                      MAP_ANONYMOUS | sharedflag, -1, 0);
    assert (addr != MAP_FAILED);
    printf("Memory mapped at %p\n", addr);

    int i, *ia = addr;
    if (fork() == 0) {
        for (i = 0; i < 10; i++)
            ia[i] = i;
    } else {
        assert (wait(NULL) > 0);
        for (i = 0; i < 10; i++)
            printf("%d ", ia[i]);
        printf("\n");
    }
    return 0;
}
```

```
int main(int ac, char *av[])
{
    size_t sz = getpagesize();
    int sharedflag = ac < 2 || strcmp(av[1], "-private")
                        ?  MAP_SHARED : MAP_PRIVATE;
    void *addr = mmap(NULL, sz, PROT_READ|PROT_WRITE,
                      MAP_ANONYMOUS | sharedflag, -1, 0);
    assert (addr != MAP_FAILED);
    sem_t *semp = (sem_t*) addr;
    assert(sem_init(semp, /* shared */1, /* initial value */ 0) == 0);

    int i, *ia = addr + sizeof(*semp);
    if (fork() == 0) {
        for (i = 0; i < 10; i++)
            ia[i] = i;
        sem_post(semp);
    } else {
        sem_wait(semp);
        for (i = 0; i < 10; i++)
            printf("%d ", ia[i]);
        printf("\n");
    }
    return 0;
}
```

mmap() & shared semaphores

```
#define PGSIZE 4096

char persistent_data[PGSIZE] __attribute__((aligned(PGSIZE)));

int
main(int ac, char *av[])
{
    int i, do_read = ac < 2 || strcmp(av[1], "-read") == 0;

    persist(".persistent_data", persistent_data,
            sizeof (persistent_data));

    if (do_read) {
        for (i = 0; persistent_data[i] != 0 && i < PGSIZE; i++)
            fputc(persistent_data[i], stdout);
    } else {
        int c;
        for (i = 0; i < PGSIZE && (c = fgetc(stdin)) != -1; i++)
            persistent_data[i] = c;

        memset(persistent_data + i, 0, PGSIZE - i); // zero rest
    }
    return 0;
}
```

mmap() & persistent variables

```
/*
 * Make variable 'variableaddr' of size 'size' persistent
 * in file 'filename'
 */
static void
persist(const char *filename, void *variableaddr, size_t size)
{
    assert (size % getpagesize() == 0);
    int fd = open(filename, O_RDWR | O_CREAT, 0666);
    assert (fd != -1);
    assert (ftruncate(fd, size) == 0);

    void *addr = mmap(variableaddr, size,
                      PROT_READ | PROT_WRITE,
                      MAP_FIXED | MAP_SHARED, fd, 0);
    assert (addr == variableaddr);
    assert (close(fd) == 0);
}
```

mmap() & persistent variables (2)