Hw 7 VE482

Wu Jiayao 517370910257

1. Page replacement algorithm

1. A clock interrupt clears the Referenced (R) bit on every clock tick.

Page	Time stamp	Present	Referenced	Modified
0	6	1	0	1
1	9	1	0	0
2	9	1	0	1
3	7	1	0	0
4	4	0	0	0

2.

Page	Time stamp	Present	Referenced	Modified
0	6	1	0	1
1	9	1	0	0
2	9	1	0	1
3	x	x	x	x
4	4	0	0	0

2. Minix 3

1. - /usr/include/minix/callnr.h
   - /usr/src/servers/pm/table.c
   - /usr/src/servers/pm/proto.h
   - /usr/src/servers/pm/signal.c
2. The order of children is not exactly the correct (sorted) order.
3.

```
int getnchpid(int n, pid_t *childpid)
{
    register struct mproc *rp;
    unsigned int children = 0;
    for (rp = &mproc[0]; rp < &mproc[NR_PROCS]; rp++)
    {
        if (rp->mp_parent == who_p)
        {
            if (children++== n)
            {
                *childpid = rp->mp_pid;
                return 1;
            }
        }
    }
    return 0;
}
```

4.

```
// Defined in /usr/src/servers/pm/proto.h
int do_getchpid(void);

// Implemented in /usr/src/servers/pm/forkexit.c
int do_getchpid(int n, pid_t* childpid) {
    int children, result;

    for (children = 0; children < n; ++children) {
        result = getnchpid(children, childpid + children);
        if (!result) break;
    }
    return children;
}
```

Compile the pm server.

5.

```
#include <unistd.h>
#include <getchpids.h>
#include <sys/types.h>
#include <stdio.h>

int main() {
    int n = 10;
    pid_t std_pid[10];
    pid_t pid
    for(int i = 0;i < 10;i++) {
        pid = fork();
        if (pid == 0) {
             sleep(1000);
            return 0;
        }
        else
        {
          std_pid[i] = pid;
        }
    }
    while(waitpid(-1,&status,0));
    pid_t childpid[10];
    int result = getchpids(10, childpid);

    printf("%d\n", result);
    if (result >= 0) {
        for(int i = 0;i < result;++i) {
            printf("During running time: %d, Collected: %d\n", std_pid[i], childpid[i]);
        }
    }
}
```

6.
   1. It has low efficiency, while finding the pid, the system needs to traverse all the process to check if certain process is the child process we need. But it is flexible, since the sub system call can be used in other system call.
   2. Maybe use a tree data structure to record the parent-child relation rather than traverse the process table.

3. Research

The ext2 or second extended file system is a file system for the Linux kernel.

The canonical implementation of ext2 is the "ext2fs" filesystem driver in the Linux kernel. Other implementations (of varying quality and completeness) exist in GNU Hurd, MINIX 3, some BSD kernels, in MiNT, and as third-party Microsoft Windows and macOS drivers.

The space in ext2 is split up into blocks. These blocks are grouped into block groups, analogous to cylinder groups in the Unix File System. There are typically thousands of blocks on a large file system. Data for any given file is typically contained within a single block group where possible. This is done to minimize the number of disk seeks when reading large amounts of contiguous data.

Each block group contains a copy of the superblock and block group descriptor table, and all block groups contain a block bitmap, an inode bitmap, an inode table, and finally the actual data blocks.

The superblock contains important information that is crucial to the booting of the operating system. Thus backup copies are made in multiple block groups in the file system. However, typically only the first copy of it, which is found at the first block of the file system, is used in the booting.

The group descriptor stores the location of the block bitmap, inode bitmap, and the start of the inode table for every block group. These, in turn, are stored in a group descriptor table.

Inode

Every file or directory is represented by an inode. The term "inode" comes from "index node" (over time, it became i-node and then inode). The inode includes data about the size, permission, ownership, and location on disk of the file or directory.

Directories

Each directory is a list of directory entries. Each directory entry associates one file name with one inode number, and consists of the inode number, the length of the file name, and the actual text of the file name. To find a file, the directory is searched front-to-back for the associated filename. For reasonable directory sizes, this is fine. But for very large directories this is inefficient, and ext3 offers a second way of storing directories (HTree) that is more efficient than just a list of filenames.

The root directory is always stored in inode number two, so that the file system code can find it at mount time. Subdirectories are implemented by storing the name of the subdirectory in the name field, and the inode number of the subdirectory in the inode field. Hard links are implemented by storing the same inode number with more than one file name. Accessing the file by either name results in the same inode number, and therefore the same data.

The special directories "." (current directory) and ".." (parent directory) are implemented by storing the names "." and ".." in the directory, and the inode number of the current and parent directories in the inode field. The only special treatment these two entries receive is that they are automatically created when any new directory is made, and they cannot be deleted.

Allocating data

When a new file or directory is created, ext2 must decide where to store the data. If the disk is mostly empty, then data can be stored almost anywhere. However, clustering the data with related data will minimize seek times and maximize performance.

ext2 attempts to allocate each new directory in the group containing its parent directory, on the theory that accesses to parent and children directories are likely to be closely related. ext2 also attempts to place files in the same group as their directory entries, because directory accesses often lead to file accesses. However, if the group is full, then the new file or new directory is placed in some other non-full group.

The data blocks needed to store directories and files can be found by looking in the data allocation bitmap. Any needed space in the inode table can be found by looking in the inode allocation bitmap.

Reference

1. Wikipedia, https://en.wikipedia.org/wiki/Ext2