VE482
Introduction to Operating Systems

HOMEWORK 7

November 28, 2021

Yihua Liu 518021910998

Ex. 1 — Page replacement algorithm

1. Explain the content of the new table entries if a clock interrupt occurs at tick 10.
   If a clock interrupt occurs at tick 10, the referenced bits will be cleared:

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

   Table 1. The system state when a clock interrupt occurs at tick 10.

2. Due to a read request to page 4 a page fault occurs at tick 10. Describe the new table entry.
   Suppose the scheduled write is not completed, the new table entries are:

Page	Time stamp	Present	Referenced	Modified
0	6	1	0	1
1	9	1	0	0
2	9	1	0	1
3	10	1	1	0
4	4	0	0	0

Table 2. The new table entries.

Ex. 2 — Minix 3

The goal of this exercise is to understand and implement system calls.

1. In which files are:

(a) the constants with number and name for the system calls?
In `/usr/src/include/minix/callnr.h`

(b) the names of the system call routines?
In `/usr/src/servers/pm/table.c`

(c) the prototypes of the system call routines?
In `/usr/src/servers/pm/proto.h`

(d) the system calls of type "signal" coded?
In `/usr/src/servers/pm/signal.c`

2. What problems arise when trying to implement a system call `int getchpids(int n, pid_t *childpid)` which "writes" the pids of up to n children of the current process into `*childpid`?
The order of the printed process identifiers of children processes might be un-determined.

3. Write a "sub-system call" `int getnchpid(int n, pid_t childpid)` which retrieves the n-th child process.
At the end of `/usr/src/servers/pm/utility.c`,

```
/*===========================================================*
 *                                do_getnchpid               *
 *===========================================================*/
int do_getnchpid(int n, pid_t *p_children)
{
  register struct mproc *rmp;

  if (p_children == NULL || n > NR_PROCS)
          return EINVAL;
  rmp = &mproc[n];
  if (rmp->mp_parent != who_p)
          return EBADSRCDST;
  *p_children = rmp->mp_pid;
  return OK;
}
```

4. Using the previous sub-system call, implement the original `getchpids` system call. The returned int value corresponds to the number of pids in `*childpid`, or -1 on an error. At the end of `/usr/src/servers/pm/utility.c`,

```
/*===========================================================*
 *                                do_getchpids               *
 *===========================================================*/
```

```
 int do_getchpids(int n, pid_t *p_children)
{
  int i;

  for (i = 0; i < n; i++) {
    if (do_getnchpid(i, p_children + i) != OK) {
      i = -1;
      break;
    }
  }

  return i;
}
```

At Line 57 of `/usr/src/include/minix/callnr.h`, add

```
#define GETCHPIDS           56
```

Change Line 70 of `/usr/src/servers/pm/table.c` into

```
        do_getchpids,       /* 56 = getchpids */
```

At the end of `/usr/src/servers/pm/proto.h`, add

```
int do_getnchpid(int n, pid_t *p_children);
int do_getchpids(int n, pid_t *p_children);
```

5. Write a short program that demonstrate the previous system calls.

```
#include <unistd.h>
#include <stdio.h>

#define CHILD_NUM 10

int main() {
  int i, n_children = 0;
  pid_t p_children[CHILD_NUM], p_children_std[CHILD_NUM];

  while (n_children <= CHILD_NUM / 2) {
    pid_t pid = fork();
    if (pid == 0)
      p_children_std[n_children++] = pid;
    else
      exit(0);
  }
```

```
  getchpid(n_children, p_children);

  for (i = 0; i < n_children; i++) {
    if (p_children_std[i] != p_children[i]) {
      fprintf(stderr, "Wrong answer %d\n", i);
    }
  }

  return 0;
}
```

6. The above strategy solves the initial problem through the introduction of a sub-system call.

   (a) What are the drawbacks and benefits of this solution?
   Drawbacks: it is not efficient to traverse all the processes.
   Benefits: it is flexible because it can be used by other system calls; it is easy to implement.

   (b) Can you think of any alternative approach? If yes, provide basic details, without any implementation.
   Use a tree to traverse the parent and child processes to improve efficiency.

Ex. 3 — Research

Write about a page on the topic of the `ext2` filesystem. Do not forget to reference your sources.

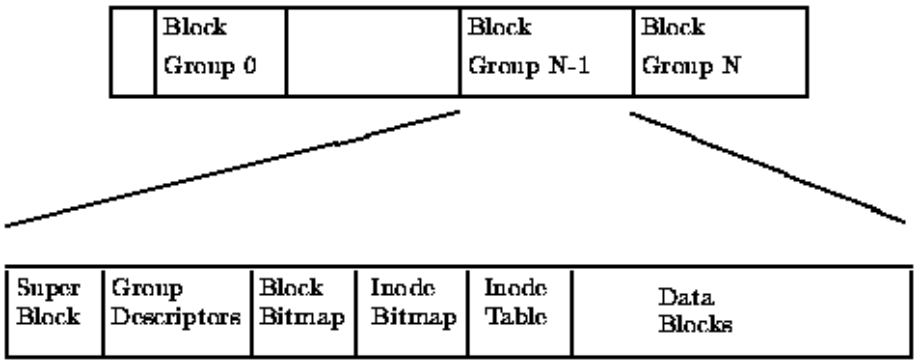

Figure 1: Physical Layout of the EXT2 File system [4].

`ext2` is a file system for the Linux kernel. Its full name is second extended file system. It was invented by Rémy Card, Theodore Ts'o and Stephen Tweedie. It is also adopted by other systems like Microsoft Windows, MacOS, and Minix 3.

The basic structure of `ext2` is blocks. "A partition, disk, file or block device" formatted with a "Second Extended Filesystem is divided into small groups of sectors called 'blocks'" [3].

Many blocks construct block groups, which contains copies of superblocks and block group descriptor tables, and the group descriptor stores "the location of the block bitmap, inode bitmap, and the start of the inode table for every block group" [5]. The block groups used to "reduce fragmentation" and minimize "the amount of head seeking when reading a large amount of consecutive data" [1].

In details, The EXT2 group descriptor is the data structure describing the block group that contains blocks bitmap, inode bitmap, inode table, free blocks count, free inodes count, and used directory count [4].

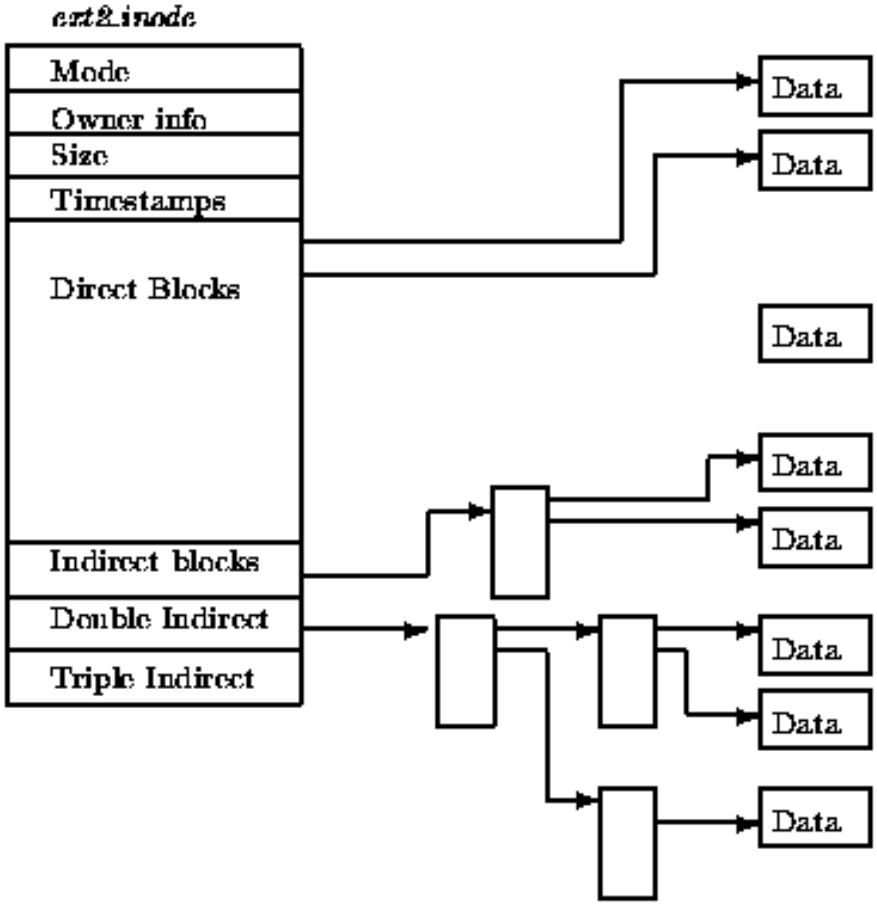

Figure 2: The EXT2 Inode [4].

The superblock, containing the configuration of the file system like the total number of inodes and blocks, are stored on disks in the format of little endian. Here inodes (whose full name is index node) are representations of files or directories.

The data structure of inodes stores the "the size, permission, ownership, and

location on disk of the file or directory" [5]. In detail, inodes contains mode, owner information, size, timestamps, and datablocks.

Directories are all like a file with an inode but also associates file names and inode numbers.

In detail, a directory is a linked list data structure. Every linked list contains "the name of the entry, the inode associated with the data of this entry, and the distance within the directory file to the next entry" [3].

Block size	Maximum file size
1024	16 GiB
2048	256 GiB
4096	2 TiB

Table 3. Block and file sizes in the second extended filesystem [2].

Ex. 4 — Simple questions

1. If a page is shared between two processes, is it possible that the page is read-only for one process and read-write for the other? Why or why not?
   No, it is not. Because the process with read-write permission to the shared page makes a copy of that page when updating that page, thus there will be two page copies. The page is no longer shared between the two process.

2. A computer provides each process with 65,536 bytes of address space divided into pages of 4096 bytes. A particular program has a text size of 32,768 bytes, a data size of 16,386 bytes, and a stack size of 15,870 bytes. Will this program fit in the address space? If the page size were 512 bytes, would it fit?
   If the page size were 4096 bytes, it would not fit.
   If the page size were 512 bytes, it would fit.

3. When both paging and segmentation are being used, first the segment descriptor is found and then the page descriptor. Does the TLB also need a two-levels lookup?
   It depends. If the page table is small, one-level TLB is small and fast enough. However, if the page table is huge and we want higher performance, two-level TLB can be introduced. A small level 1 TLB is extremely fast and small while a larger level 2 TLB is relatively slower.

References

[1] The kernel development community. *The Linux Kernel documentation*. Version 5.16.0-rc2. Free Software Foundation, Inc. 2021. URL: `https://www.kernel.org/doc/html/latest/index.html`.

[2] Wolfgang Mauerer. *Professional Linux Kernel Architecture*. John Wiley & Sons, 2010. ISBN: 9781118079911.

[3] Dave Poirier. *The Second Extended File System*. 2019. URL: `https://www.nongnu.org/ext2-doc/ext2.html`.

[4] David A. Rusling. *The Linux Kernel*. Version DRAFT, 0.1-10(30). Apr. 1997. URL: http://www.science.unitn.it/~fiorella/guidelinux/tlk/.

[5] Wikipedia contributors. *Ext2 — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-November-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Ext2&oldid=1057194173.