VE482 Homework 7

Due: Nov.24

Name: Wu Qinhang

ID: 518370910041

Email: william_wu@sjtu.edu.cn

Ex1 Page Replacement Algorithm

Explain the content of the new table entries if a clock interrupt occurs at tick 10.

During the clock interrupt, the reference bit on every clock tick will be cleared. Therefore, the new table entries should look like:

```
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
```

Due to a read request to page 4 a page fault occurs at tick 10. Describe the new table entry.

Page 4 is not exist in the physical memory, so the page replacement algorithm (WSClock) need to find an existed page to replace. It locates Page 3.

The new table entries should look like:

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

Ex2 Minix 3 System Call

1. System call implementation: [1]

   a) The constants with number and name for the system call: `/include/minix/callnr.h`

```
#define NCALLS     114  /* number of system calls allowed */

/* In case it isn't obvious enough: this list is sorted numerically.
*/
#define EXIT       1
#define FORK       2
#define READ       3
// ...
```

b) name of system call routine: /servers/*/table.c

```
/* This file contains the table used to map system call numbers
onto the
 * routines that perform them.
 *
 * Created (MFS based):
 *   February 2010 (Evgeniy Ivanov)
 */
int (*fs_call_vec[])(void) = {
    no_sys,             /* 0   not used */
    no_sys,             /* 1   */       /* Was: fs_getnode */
    fs_putnode,         /* 2   */
    // ...
```

c) prototypes of the system call routines: /servers/*/proto.h

```
/* Function prototypes. */

struct vmproc;
struct stat;
struct memory;
struct vir_region;
struct phys_region;

/* alloc.c */
void mem_sanitycheck(char *file, int line);
phys_clicks alloc_mem(phys_clicks clicks, u32_t flags);
void memstats(int *nodes, int *pages, int *largest);
void printmemstats(void);
```

d) the system calls of type "signal" coded: /servers/pm/signal.c

```
/* This file handles signals, which are asynchronous events and are
generally
 * a messy and unpleasant business.  Signals can be generated by
the KILL
 * system call, or from the keyboard (SIGINT) or from the clock
(SIGALRM).
 * In all cases control eventually passes to check_sig() to see
which processes
 * can be signaled.  The actual signaling is done by sig_proc().
 *
 * The entry points into this file are:
```

```
 *   do_sigaction:  perform the SIGACTION system call
 *   do_sigpending: perform the SIGPENDING system call
 *   do_sigprocmask:  perform the SIGPROCMASK system call
 *   do_sigreturn:  perform the SIGRETURN system call
 *   do_sigsuspend: perform the SIGSUSPEND system call
 *   do_kill:   perform the KILL system call
 *   do_pause:    perform the PAUSE system call
 *   process_ksig:  process a signal an behalf of the kernel
 *   sig_proc:    interrupt or terminate a signaled process
 *   check_sig:   check which processes to signal with sig_proc()
 *   check_pending: check if a pending signal can now be delivered
 *   restart_sigs:  restart signal work after finishing a VFS call
 */

static void unpause(struct mproc *rmp);
static int sig_send(struct mproc *rmp, int signo);
static void sig_proc_exit(struct mproc *rmp, int signo);
// ...
```

2. there may exist some memory accessing issues since `getchpids` is called from user space.

3.
```
#include <unistd.h>
#include "mproc.h"

#define OK 0
/**
 * @brief helper function that retrieves the n^{th} child process
 *
 * @param n next_child
 * @param childpid array that stores the pid of the child processes
 * @return int
 */
int getnchpid(int n, pid_t *childpid){
  register struct mproc *rmc; // pointer to child
  if (childpid == NULL) return -1;
  if (n > NR_PROCS) return -1;
  rmc = &mproc[n];
  if (rmc->mp_parent != who_p) return -1;
  *childpid = rmc->mp_pid;

  return OK;
  // on error return -1
}
```

4.
```
#include <unistd.h>
#include "mproc.h"

#define OK 0
/**
 * @brief write the pids of up to n children of the current process
into *childpid
```

```
 *
 * @param n
 * @param childpid
 * @return int
 */
int getchpids(int n, pid_t *childpid){
  int index;
  for (index = 0; index < n; index++){
    if (getnchpid(index, childpid+index) != OK){
      index = -1;
      break;
    }
  }

  return index;
}
```

5.
```
#include <unistd.h>
#include <stdio.h>
#include <assert.h>
int main(){
  pid_t childpid[10];
  int childNum = 0;
  pid_t childpidRef[10];

  while(childNum <= 5){
    pid_t pid = fork();
    if (pid == 0){
      childpidRef[childNum++] = pid;
    }
    else{
      exit (0);
    }
  }

  getchpid(childNum, childpid);

  for(int i=0;i<childNum;i++){
    assert(childpidRef[i] == childpid[i]);
  }

  return 0;
}
```

6. a) Drawbacks: performance is low since it requires several calls of the sub-system call; benefits: safe, and easy implementation

   b) alternative approach: pass the whole array and let the sub-system call handle the whole job.

Ex3 Research on ext2 filesystem [2] [3]

ext2, known for the **second extended file system**, is a commercial-grade file system designed for Linux kernel. Nowadays, ext2 has become the default filesystem in many distributed Linux system.

The space in ext2 is designed to be separated into `blocks`. They are furthur grouped into block groups. Usually data is contained within a single block group so that the disk seek can be reduced as much as it can. Each block group contains a replica of the superblock, group descriptor table, and a map into the actual data block.

`inode` (index node) is another important concept since every file or directory in ext2 structure is represented by it. It includes the information about the size, permission, ownership, and the specific location in the disk (Figure 1).

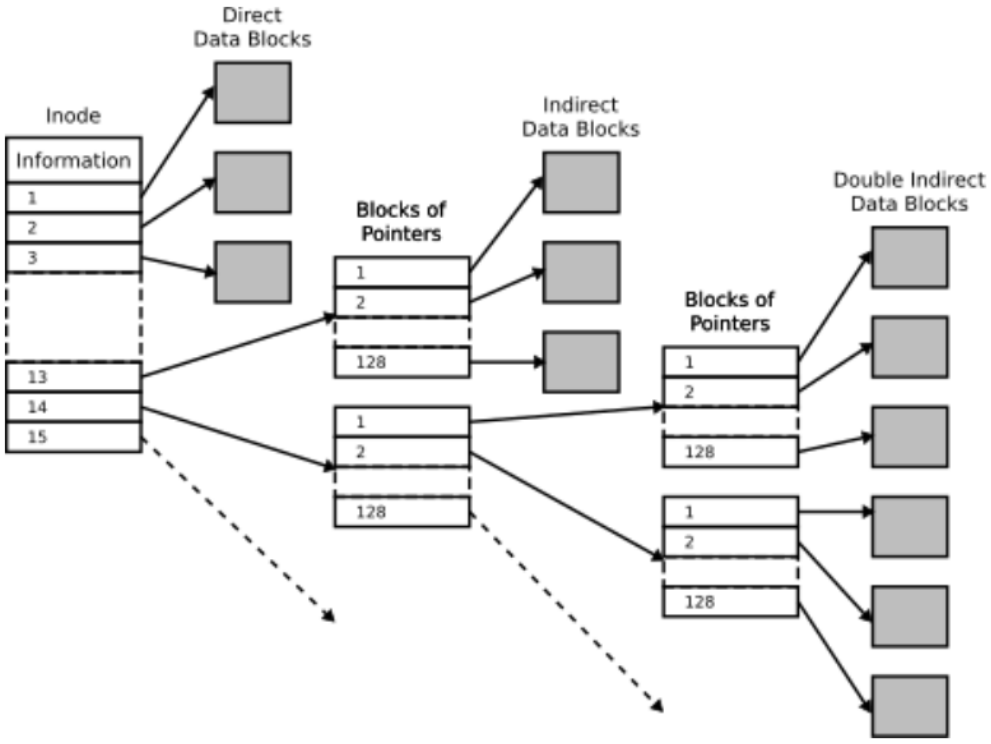

Figure 1. inode structure of ext2 [2]

It is said that:

> There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly indirect block and a pointer to a trebly indirect block.

The directory consists of a list of directory entries which is associated with a corresponding inode number. Two special directory exists: `.` for current directory and `..` for parent directory. They are implemented by storing two names directly into the directory, which are automatically created.

The data allocation is also controlled by ext2 strictly. New directory usually is allocated in the group that contains the parent directory. If the group is full, a new directory is seeked to place the data.

Ex4 Page

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. Because even if a page is shared between two processes, any updates from a read-write will cause a copy for the page, so that it will no longer be shared among them.

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?

For a page of 4096 bytes, **no.**

For a page of 512 bytes, **yes.**

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?

No. It is not necessary.

Reference

1. "ECS150 Fall 2003," *Usfca.edu* , 2020. https://www.cs.usfca.edu/~sjengle/ucdavis/ecs150-f03/syscall.html (accessed Nov. 24, 2020). ↩

2. Wikipedia Contributors, "ext2," *Wikipedia* , Oct. 30, 2020. https://en.wikipedia.org/wiki/Ext2#:~:text=The%20ext2%20or%20second%20extended,extended%20file%20system%20(ext) . (accessed Nov. 24, 2020). ↩

3. "The Second Extended File System," *Nongnu.org* , 2019. https://www.nongnu.org/ext2-doc/ext2.html (accessed Nov. 24, 2020). ↩