



上海工程技术大学

实验报告

实验四 内存管理

1. 实验要求

(1) 通过在 Linux 环境下对内存管理的基本操作了解 Linux 如何对内存及虚拟内存进行管理；

(2) 掌握可变分区管理内存的方法。

2. 实验内容

1) 验证实验：

- ① Linux 命令 FREE 显示内存状态情况，观察结果并分析；
- ② 用 VMSTAT 命令监视虚拟内存使用情况，观察结果并分析；
- ③ 使用 PS 和 KILL 命令回收内存，观察结果并分析（例如：可以打开两个终端，一个终端先运行无限循环程序，然后再另一个终端 KILL 该进程）；
- ④ 使用 SYNC 命令将内存缓冲区的数据写入磁盘；
- ⑤ 观察 ULIMIT 命令结果，写出用法和作用。

2) 编程实验，编写程序实现采用可变分区方法管理内存：

- ① 设计实现用来记录主存使用情况的数据结构：已分区表和空闲分区表或链表；
- ② 在设计好的数据结构上设计实现循环首次适应算法；
- ③ 在设计好的数据结构上设计实现主存回收算法。其中，若回收的分区有上邻空闲分区和（或）下邻空闲分区，要求合并为一个空闲分区登记在空闲分区表的一个表项里。

3. 实验结果

1) 验证实验：

- ① Linux 命令 FREE 显示内存状态情况，观察结果并分析；

```
ldd@ldd-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ldd@ldd-virtual-machine:~$ free
              总计              已用              空闲              共享              缓冲/缓存              可用
内存:         2017292         789620         885564         13124         342108         1051592
交换:          969960         497912         472048

ldd@ldd-virtual-machine:~$ free -m
              总计              已用              空闲              共享              缓冲/缓存              可用
内存:           1970           1054             89              6             826           727
交换:            947             141            805
```

② 用 VMSTAT 命令监视虚拟内存使用情况，观察结果并分析；

```
ldd@ldd-virtual-machine:~$ vmstat
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
r  b 交换 空闲 缓冲 缓存  si  so  bi  bo  in  cs  us  sy  id  wa  st
0  0 441848 304660 77772 631684  4  21  127  177  225  514  8 10 81  0  0
```

③ 使用 PS 和 KILL 命令回收内存，观察结果并分析（例如：可以打开两个终端，一个终端先运行无限循环程序，然后再另一个终端 KILL 该进程）；

```
ldd@ldd-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ldd@ldd-virtual-machine:~$ ps -elf
F S UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root          1        0  0  80   0 - 56435 -          02:38 ?           00:00:05 /sb
1 S root          2        0  0  80   0 -    0 -          02:38 ?           00:00:00 [kt
1 I root          4        2  0  60 -20 -    0 -          02:38 ?           00:00:00 [kw
1 I root          6        2  0  60 -20 -    0 -          02:38 ?           00:00:00 [mm
1 S root          7        2  0  80   0 -    0 -          02:38 ?           00:00:02 [ks

0 S ldd          23723   23610  0  80   0 - 6090 wait  11:46 pts/0           00:00:00 bas
0 R ldd          23731   23723  0  80   0 - 10358 -    11:46 pts/0           00:00:00 ps

ldd@ldd-virtual-machine:~$ kill -9 23723
ldd@ldd-virtual-machine:~$
```

④ 使用 SYNC 命令将内存缓冲区的数据写入磁盘；

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ldd@ldd-virtual-machine:~$ sync
ldd@ldd-virtual-machine:~$
```

```
ltd@ltd-virtual-machine: ~
CPU: 0.7% Tasks: 134, 300 thr: 1 running
Mem: 1.06G/1.92G Load average: 0.16 0.10 0.03
Swap: 1.60M/9.47M Uptime: 10:29:51

PID USER PR1 NI VIRT RES SHR S CPU% MEM% TIME+ Command
12792 ldd 20 0 33200 3716 3656 S 0.7 0.2 0:00.12 httpd
28179 ldd 20 0 8630 4866 13456 S 0.7 2.2 0:05.97 /usr/lib/gnome-terminal/gnome-terminal-server
23321 ldd 20 0 29190 1680 55180 S 0.0 4.1 0:46.89 /usr/bin/gnome-shell
23184 ldd 20 0 4130 46960 14676 S 0.0 2.2 0:26.04 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthori
23196 ldd 20 0 4130 46960 14676 S 0.0 2.2 0:02.97 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthori
1631 gdm 20 0 6380 9548 7276 S 0.0 0.5 0:05.77 /usr/lib/gnome-settings-daemon/gsd-color
1 root 20 0 2200 7352 5060 S 0.0 0.4 0:06.16 /sbin/init splash
326 root 19 0 1810 17448 16728 S 0.0 0.9 0:02.52 /lib/systemd/systemd-journald
348 root 20 0 36632 4276 2528 S 0.0 0.2 0:00.54 /lib/systemd/systemd-udevd
478 systemd-r 20 0 78872 4272 3844 S 0.0 0.2 0:01.19 /lib/systemd/systemd-resolved
523 systemd-t 20 0 1420 3868 3816 S 0.0 0.2 0:00.00 /lib/systemd/systemd-timesyncd
481 systemd-t 20 0 1420 3868 3816 S 0.0 0.2 0:00.12 /lib/systemd/systemd-timesyncd
776 root 20 0 4240 5176 4736 S 0.0 0.3 0:00.00 /usr/sbin/ModemManager --filter-policy=strict
784 root 20 0 4240 5176 4736 S 0.0 0.3 0:00.00 /usr/sbin/ModemManager --filter-policy=strict
745 root 20 0 4240 5176 4736 S 0.0 0.3 0:00.09 /usr/sbin/ModemManager --filter-policy=strict
769 syslog 20 0 2560 1992 1820 S 0.0 0.1 0:00.12 /usr/sbin/rsyslogd -n
770 syslog 20 0 2560 1992 1820 S 0.0 0.1 0:00.04 /usr/sbin/rsyslogd -n
771 syslog 20 0 2560 1992 1820 S 0.0 0.1 0:00.11 /usr/sbin/rsyslogd -n
747 syslog 20 0 2560 1992 1820 S 0.0 0.1 0:00.33 /usr/sbin/rsyslogd -n
772 root 20 0 2640 5720 5080 S 0.0 0.3 0:00.51 /usr/lib/AccountsService/accounts-daemon
783 root 20 0 2640 5720 5080 S 0.0 0.3 0:00.07 /usr/lib/AccountsService/accounts-daemon
751 root 20 0 2640 5720 5080 S 0.0 0.3 0:00.68 /usr/lib/AccountsService/accounts-daemon
760 root 20 0 78736 3928 3576 S 0.0 0.2 0:00.44 /lib/systemd/systemd-logind
764 messagebu 20 0 51864 5356 3152 S 0.0 0.3 0:03.16 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile
822 root 20 0 5510 11080 8912 S 0.0 0.5 0:00.40 /usr/sbin/NetworkManager --no-daemon
827 root 20 0 5510 11080 8912 S 0.0 0.5 0:00.22 /usr/sbin/NetworkManager --no-daemon
782 root 20 0 5510 11080 8912 S 0.0 0.5 0:01.73 /usr/sbin/NetworkManager --no-daemon
785 root 20 0 4560 480 480 S 0.0 0.0 0:00.72 /usr/sbin/acpid
787 avahi 20 0 47264 2888 2612 S 0.0 0.1 0:00.44 avahi-daemon: running [ltd-virtual-machine.local]
788 root 20 0 33080 2172 2076 S 0.0 0.1 0:00.09 /usr/sbin/cron -f
789 root 20 0 45240 1648 1616 S 0.0 0.1 0:00.12 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
790 avahi 20 0 47884 32 0 S 0.0 0.0 0:00.00 avahi-daemon: chroot helper
```

⑤ 观察 ULIMIT 命令结果，写出用法和作用。

```
ltd@ltd-virtual-machine: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

ltd@ltd-virtual-machine:~$ ulimit
unlimited
ltd@ltd-virtual-machine:~$ ulimit -a
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
scheduling priority (-e) 0
file size (blocks, -f) unlimited
pending signals (-i) 7703
max locked memory (kbytes, -l) 65536
max memory size (kbytes, -m) unlimited
open files (-n) 1024
pipe size (512 bytes, -p) 8
POSIX message queues (bytes, -q) 819200
real-time priority (-r) 0
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 7703
virtual memory (kbytes, -v) unlimited
file locks (-x) unlimited
ltd@ltd-virtual-machine:~$ ulimit -H
unlimited
ltd@ltd-virtual-machine:~$ ulimit -S
unlimited
```

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ltd@ltd-virtual-machine:~$ ulimit --help
ulimit: ulimit [-SHabcdefiklmnpqrstuvxPT] [限制]
修改 shell 资源限制。

在允许此类控制的系统上，提供对于 shell 及其创建的进程所可用的
资源的控制。

选项：
-S      使用软 ('soft') 资源限制
-H      使用硬 ('hard') 资源限制
-a      所有当前限制都被报告
-b      套接字缓存尺寸
-c      创建的核文件的最大尺寸
-d      一个进程的数据区的最大尺寸
-e      最高的调度优先级 ('nice')
-f      有 shell 及其子进程可以写的最大文件尺寸
-i      最多的可以挂起的信号数
-k      分配给此进程的最大 kqueue 数量
-l      一个进程可以锁定的最大内存尺寸
-m      最大的内存进驻尺寸
-n      最多的打开的文件描述符个数
-p      管道缓冲区尺寸
-q      POSIX 信息队列的最大字节数
-r      实时调度的最大优先级
-s      最大栈尺寸
-t      最大的CPU时间，以秒为单位
-u      最大用户进程数
-v      虚拟内存尺寸
-x      最大的文件锁数量
-P      最大伪终端数量
-T      最大线程数量

并非所有选项在所有系统上可用。

如果提供了 LIMIT 变量，则它为指定资源的新的值；特别的 LIMIT 值为
'soft'、'hard'和'unlimited'，分别表示当前的软限制，硬限制和无限制。
否则打印指定资源的当前限制值，不带选项则假定为 -f

取值都是 1024 字节为单位，除了 -t 以秒为单位，-p 以 512 字节递增，
-u 为无范围的进程数量
```

编程实验核心代码：

```
void show()
{
    int flag = 0;//用来记录分区序号
    Node* p = first;
    p->data.num = 0;
    p->data.address = 0;
    p->data.length = 40;
    p->data.state = 1;
    sort();
    printf("\n\t\t 主存空间分配情况\n");
    printf("分区序号\t 起始地址\t 分区大小\t 分区状态\n\n");
    while (p)
    {
        printf("%d\t\t%d\t\t%d", p->data.num, p->data.address, p->data.length);
        if (p->data.state == 0)
            printf("\t\t 空闲\n\n");
        else
            printf("\t\t 已分配\n\n");
        p = p->next;
    }
}
```

```

}
Status First_fit(int request) { //为申请作业开辟新空间且初始化
    Node* p = first->next;
    LinkList temp = (LinkList)malloc(sizeof(Node));
    temp->data.length = request;
    temp->data.state = 1;
    p->data.num = 1;
    while (p)
    {
        if ((p->data.state == 0) && (p->data.length == request))
            { //有大小恰好合适的空闲块
                p->data.state = 1;
                return OK;
                break;
            }
        else if ((p->data.state == 0) && (p->data.length > request))
            { //有空闲块能满足需求且有剩余
                temp->prior = p->prior;
                temp->next = p;
                temp->data.address = p->data.address;
                temp->data.num = p->data.num;
                p->prior->next = temp;
                p->prior = temp;
                p->data.address = temp->data.address + temp->data.length;
                p->data.length -= request;
                p->data.num += 1;
                return OK;
                break;
            }
        p = p->next;
    }
    return ERROR;
}

```

实验结果：


```

    本程序仅为使用首次适应算法（FF）实现主存空间的分配，BF、WF算法未进行实现
请输入1调用内存分配算法:1
使用首次适应算法:

分区序号      主存空间分配情况
起始地址      分区大小      分区状态
0              0              40          已分配
1              40              600         空闲

    1: 分配内存    2: 回收内存    0: 退出
请输入您的操作: 1
请输入申请分配的主存大小(单位:KB):2000
    $$内存不足, 分配失败!$$
    主存空间分配情况
分区序号      起始地址      分区大小      分区状态
0              0              40          已分配
1              40              600         空闲

    1: 分配内存    2: 回收内存    0: 退出
请输入您的操作: 1
请输入申请分配的主存大小(单位:KB):200
    $$分配成功! $$
    主存空间分配情况
请输入申请分配的主存大小(单位:KB):200
    $$分配成功! $$
    主存空间分配情况
分区序号      起始地址      分区大小      分区状态
0              0              40          已分配
1              40              200         已分配
2              240             400         空闲

    1: 分配内存    2: 回收内存    0: 退出
请输入您的操作: 2
请输入您要释放的分区号: 1
    $$回收成功$$
    主存空间分配情况
分区序号      起始地址      分区大小      分区状态
0              0              40          已分配
1              40              200         空闲
2              240             400         空闲

    1: 分配内存    2: 回收内存    0: 退出
请输入您的操作: 1
请输入申请分配的主存大小(单位:KB):10
    $$分配成功! $$
    主存空间分配情况
分区序号      起始地址      分区大小      分区状态

```

4. 实验小结