exec 和 堆空间管理

同济大学计算机系　操作系统作业　　2023-12-7

学号　　姓名

一、　这个程序的输出是什么?

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

int main1( )    // tryExec.c
{
   char* argv[4];
   argv[0] = "showCmdParam";
   argv[1] = "arg1";
   argv[2] = "arg2";
   argv[3] = 0;

   if ( fork( ) ==0 )
         ;
   else{
      execv(argv[0] , argv) ;
      printf("Over!\n") ;
   }
   exit(0);
}
```

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

int main1(int argc, char *argv[])   // showCmdParam.c
{
   int i;

   printf("The command parameter of showCmdParam\n");

   for(i = 0; i < argc; i++)
      printf("argv[%d]:\t%s\n", i, argv[i]);

   exit(0);
}
```

解:

我们通读上面的程序，可以知道程序的输出应该是:

输出如下：

The command parameter of showCmdParam (这里是换行)

argv[0]: showCmdParam (这里是换行)

argv[1]: arg1 (这里是换行)

argv[2]: arg2 (这里是换行)

Over! (这里是换行)

二、现运行进程 PA，1 页代码，1 页数据，没有只读数据 和 bss，1 页堆栈。代码段起始 0x401000。进程依次执行下列动态内存分配释放操作。

char *p1= malloc(4) ;	(1) 指针 p1 的值是多少 ?
	指针 p1 的值是 0X403010
char *p2= malloc(4) ;	(2) 指针 p2 的值是多少 ?
	指针 p1 的值是 0X403020
char *p3= malloc(32) ;	(3) 指针 p3 的值是多少 ?
free(p2);	指针 p1 的值是 0X403030
char *p4= malloc(8) ;	(4) 指针 p4 的值是多少 ?
free(p1);	指针 p1 的值是 0X403020
char *p5= malloc(8) ;	(5) 指针 p5 的值是多少 ?

指针 p1 的值是 0X403010

（6）情景分析

解：我们对上面的过程有下面的情景分析：

首先，代码段的起址地址为 0x401000，长度为 1 页。

数据段的起址地址为 0x402000，长度为 1 页。

堆栈段的起址地址为 0x403000，

之后我们为数据段追加 12K 的数据，即 malloc_head=0x403000，malloc_end=0x406000，其中 malloc_head 的位置进行初始化时会写入一个 flist 结构，它的大小 8 字节。我们考察 p1 的内存片的构成，其中 flist 的大小为 8B，need 的大小是 4B，则大小应为 8+4=12B，对齐之后的大小为 16B；同时 p1 指向 malloc 的返回值。这个返回值指向的是 flist 的尾部，并不是内存片；因此 p1 指向的地址为 0x403010。

之后我们向内存片 1 的后面追加 flist 结构和 8B，此时 p2 指向的地址是 0x403020，p3 指向的地址是 0x403030。这时候内存片 2 会被释放。

内存片 1、3 之间大小为 8+8=16B，满足 p4，故 p4 指向的地址为 0x403020。

之后 malloc_head 结构的 flist 与内存片 4 之间的区域大小为 8+8=16B 满足 p5，故 p5 指向的地址为 0x403010。

(7) 画最终的堆结构图

堆结构图应该为：

P5：0X403010
P4：0X403020
P3：0X403030

三、简述 tryExec 进程创建子进程的过程

解：

（1）首先父进程 tryExec 进程会为应用程序准备相应的命令行参数*argv[n]。

（2）之后会执行 fork()函数，进行系统调用创建子进程。

（3）之后父进程和子进程会一起执行 tryExec 程序，在选择结构的代码：if(fork()) else 的选择结构，在这个选择结构中会转到 else 这个分支，之后会调用 execv()。

（4）在 execv()函数中，会通过内联的汇编代码执行 exec 函数，进行系统调用刷新子进程的用户空间，并装入新的进程图像。

（5）在从 exec()函数中返回后，由于子进程处于用户态，之后会从其 main()函数的第一条指令开始执行。

四、简述程序 showCmdParam 的加载过程

解：

如果是使用 tryExec 进程创建的进程，加载过程如下：

（1）首先 tryExec 进程会准备命令行参数*argv[n]。

（2）之后会执行 fork()函数，进行系统调用创建子进程。

（3） 之后创建的 showCmdParam 子进程在 fork()函数的执行结束后，会在父进程中的选择结构中执行 else 分支，调用并执行 execv()函数。

（4）之后钩子函数会计算清点得到 argc，并通过内联的汇编代码进入 11 号系统调用，并

将 argv[0]下的应用程序的图像装入子进程。
（5）最后在 showCmdParam 进程的 exec 系统调用执行完毕后，会返回函数，由于处于用户态，所以会从 main()的第一句代码开始执行。

如果是 shell 创建的进程
（1）首先会新建 gcc 进程，该进程是 showCmdParam 进程，也是 shell 进程的子进程。
（2）在 showCmdParam 进程的 fork()函数返回的时候，会执行 shell 进程选择结构中的 else 分支语句。
（3）之后会执行 exec(“gcc”,arg1…)的系统调用：会先清除线性地址空间中的 shell 进程图像，随后会装填子进程的代码段、MemoryDescriptor、虚实地址表、可交换部分、数据段等。
（4）之后要创建 showCmdParam 进程的 main()函数栈帧，压入 argv[N]参数，并执行寄存器清 0 的操作等等。
（5）最后新进程会在 exec()函数返回之后执行 gcc 任务。

五、简述子进程 showCmdParam 进程，的终止过程
（1）首先会传递终止码 status 作为参数，并执行钩子函数 exit(status)。
（2）之后 exit()函数通过内联的汇编代码进入 1 号系统，并调用 Sys_Rexit()函数。
（3）在 Sys_Rexit()函数中会调用其内核函数 Exit()完成状态变迁。
（4）Exit()函数需要完成以下的主要工作，复制 User 结构到盘交换区，并释放各类资源，之后会唤醒父进程，并回收当前进程 PCB。之后会将当前进程的子进程的 ppid 改为 1#，唤醒 1#进程；最后当前进程会放弃 CPU。

二、三、四，不必过于详细。要求掌握相关 PPT 的标题和主干步骤。