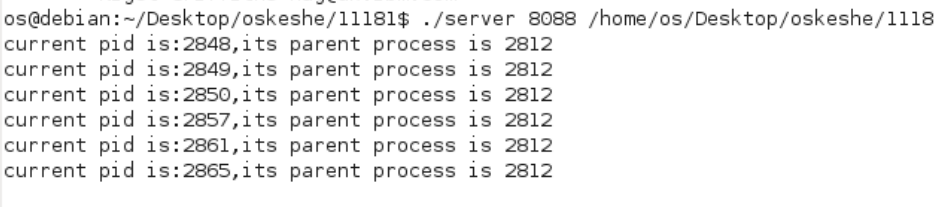
操作系统第二次课程设计

第一题，实现webserver多进程并发处理。即有客户端访问webserver时，webserver接收到且没有异常现象，则通过accept方法接收客户端请求，子进程也在这里创建，用于客户端访问。代码截图如下



其中第11行，是接收客户端请求后创建子进程，然后web的操作在子进程里执行，每次输出子进程和父进程，以判断程序确实在子进程里执行的。只输出子进程和父进程即可，没有计算web函数执行时间，因为进程不一样且没有信号量确保子进程不会冲突执行，无法计算每个进程的web执行时间。

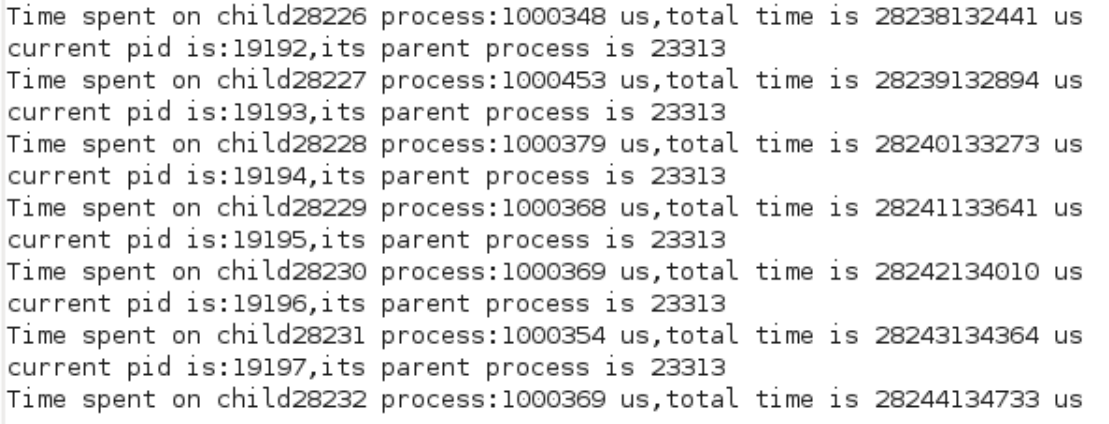
截图如下所示，每个我点击了一次刷新，所以有六个进程。一次访问http会创建三个子进程如下图所示（只显示了一部分），第一个子进程加载html，第二个加载jpg，第三个加载ico。每个子进程在执行相应的操作，而所有子进程的父进程都是一个，由此可见webserver是多进程并发执行的。



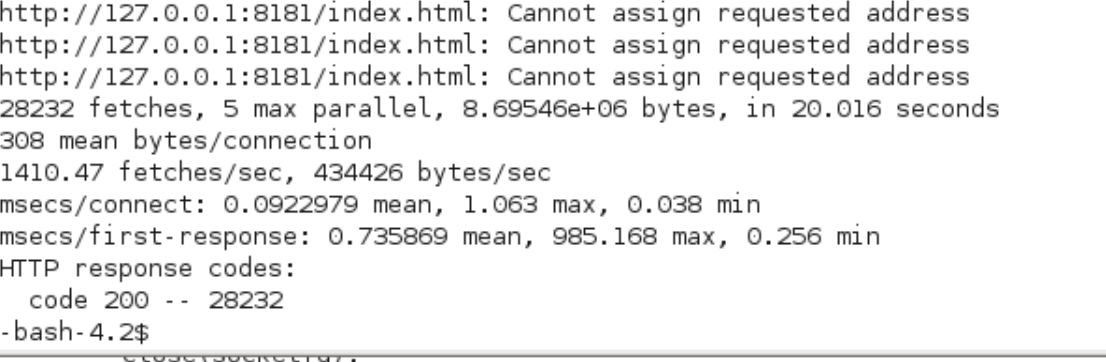
第二题，使用信号量、共享内存等系统接口函数，来统计每个子进程的 消耗时间以及所有子进程消耗时间之和。此题需要用信号量来限制每个进程用于精准计算时间，因为在多进程并发中，gettimeofday方法的值需要在每个进程中被保护，不能不同进程混用返回值，所以采用的方法就是用sem\_wait()，此方法是将每个执行中的进程的资源保护起来，只允许一个进程进入临界区，从而不会造成冲突。



信号量的作用是确保每次算时间能算准确，而共享内存的作用是能算出所有的子进程执行时间之和，如下图所示，输出中加上了所有子进程执行的时间。其计算方法也在上图代码中，281行，memptr是把所有进程时间加一起的，先在父进程中创建共享内存，从而父进程的所有子进程此资源是共用的，任何进程都可修改，就能计算总的进程时间了，如下图所示。



第三题，使用http\_load来测试当前设计的多进程WebServer服务性能， 根据测试结果来分析其比单进程Web服务性能提高的原因。同时结合题目2，来分析当前多进程WebServer的性能瓶颈在何处？是否还能够继续 提高此WebServer服务的性能？



上图是我测试的http\_load多进程模式下的结果。在20.016秒内最大启动了5个客户端，完成28232次抓取，共传输8.69MB的数据，没有完成50次网页抓取。每次连接平均传输308个字节。每秒平均完成1410.47次网页传输，每秒传输434426字节的数据。性能和单进程相比有明显提高。原因是，多进程模型中每当接收到用户请求，系统不是自己去执行请求操作，并让后面的请求等着，而是把请求通过子进程来执行，从而把任务分配到其他地方，也就大大减小了请求的等待时间，从而能处理更多任务。Webserver的瓶颈目前在于web操作的执行方式，是代码实现层面的瓶颈，还能继续优化，方法是优化web的实现方式。还可以通过线程来执行webserver的服务性能，因为线程执行速度比进程更快。

下面将我遇到的困难和解决办法及原理描述：

首先是在子进程中创建了进程，但我没有在父进程里写close（sockeid）的时候，web只能返回html的操作请求结果，后面的jpg和ico全都没有，后来我了解到socket操作返回的是输出流，也就是说子进程执行时，如果我不在父进程关闭接收到的socketid，后面的jpg和ico的请求数据就会丢失，其对应的是web中bind操作返回值为-1，通过errno检查得到错误号2号，是unknown error。这个问题困扰了我好久。

第二个困难是对信号量的理解，我一开始以为是在子进程开始处添加信号量，也就是把web放在sem\_wait()和sem\_post()中间，但这样用http\_load看却还是按照单进程执行，原因是资源被锁死了，只允许一个进程进入临界区，其他进程继续等着，这和单进程没有区别。后来我多了解了信号量是干嘛的，我才意识到是为了不让各个进程的计算时间的数据冲突，所以应该加在timeuse和memptr那里。

——2021.11.24

附 源代码（第一题的包含在二三题里面）

//semaphoreposix.c 文件

//使用下面的命令来编译下面代码

// gcc -std=gnu99 -Wall -g -o server1118 server1118.c -lrt -lpthread

#include <sys/mman.h>

#include <semaphore.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <fcntl.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <time.h>

#define NUM 10

#define SEM\_NAME "server.c"

#define SHM\_NAME "server.c"

#define VERSION 23

#define BUFSIZE 8096

#define ERROR      42

#define LOG        44

#define FORBIDDEN 403

#define NOTFOUND  404

#ifndef SIGCLD

#define SIGCLD SIGCHLD

#endif

struct {

  char \*ext;

  char \*filetype;

} extensions [] = {

  {"gif", "image/gif" },

  {"jpg", "image/jpg" },

  {"jpeg","image/jpeg"},

  {"png", "image/png" },

  {"ico", "image/ico" },

  {"zip", "image/zip" },

  {"gz",  "image/gz"  },

  {"tar", "image/tar" },

  {"htm", "text/html" },

  {"html","text/html" },

  {0,0} };

int shm\_open(const char \*name, int oflag, mode\_t mode);

int shm\_unlink(const char \*name);

void \*mmap(void \*start, size\_t len, int prot, int flags, int fd, off\_t offset);

/\* 日志函数，将运行过程中的提示信息记录到webserver.log 文件中\*/

void logger(int type, char \*s1, char \*s2, int socket\_fd)

{

  int fd ;

  struct timeval t1,t2;

  double timeuse;

  char logbuffer[BUFSIZE\*2];

  time\_t timep;

  struct tm \*p;

  time(&timep);

  p = gmtime(&timep);

  /\*根据消息类型，将消息放入logbuffer缓存，或直接将消息通过socket通道返回给客户端\*/

  gettimeofday(&t1,NULL);

  switch (type) {

  case ERROR:

    (void)sprintf(logbuffer,"date:%d\\%d\\%d %d:%d:%d\nERROR: %s:%s Errno=%d exiting pid=%d",(1900+p->tm\_year),(1+p->tm\_mon),p->tm\_mday,p->tm\_hour+8,p->tm\_min,p->tm\_sec,s1, s2, errno,getpid());

    break;

  case FORBIDDEN:

    (void)write(socket\_fd, "HTTP/1.1 403 Forbidden\nContent-Length: 185\nConnection: close\nContent-Type: text/html\n\n<html><head>\n<title>403 Forbidden</title>\n</head><body>\n<h1>Forbidden</h1>\n The requested URL, file type or operation is not allowed on this simple static file webserver.\n</body></html>\n",271);

    (void)sprintf(logbuffer,"date:%d\\%d\\%d %d:%d:%d\nFORBIDDEN: %s:%s",(1900+p->tm\_year),(1+p->tm\_mon),p->tm\_mday,p->tm\_hour+8,p->tm\_min,p->tm\_sec,s1, s2);

    break;

  case NOTFOUND:

    (void)write(socket\_fd,  "HTTP/1.1  404  Not  Found\nContent-Length:  136\nConnection: close\nContent-Type:  text/html\n\n<html><head>\n<title>404  Not Found</title>\n</head><body>\n<h1>Not  Found</h1>\nThe  requested  URL  was  not  found  on  this server.\n</body></html>\n",224);

    (void)sprintf(logbuffer,"date:%d\\%d\\%d %d:%d:%d\nNOT FOUND: %s:%s",(1900+p->tm\_year),(1+p->tm\_mon),p->tm\_mday,p->tm\_hour+8,p->tm\_min,p->tm\_sec,s1, s2);

    break;

  case LOG:

    (void)sprintf(logbuffer,"date:%d\\%d\\%d %d:%d:%d\nINFO: %s:%s:%d",(1900+p->tm\_year),(1+p->tm\_mon),p->tm\_mday,p->tm\_hour+8,p->tm\_min,p->tm\_sec,s1, s2,socket\_fd);

    break;

  }

  gettimeofday(&t2,NULL);

  timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

  /\*printf("Time spent on putting in loggbuffer cache：%lf\n",timeuse);\*/

  /\* 将logbuffer缓存中的消息存入webserver.log文件\*/

  gettimeofday(&t1,NULL);

  if((fd = open("webserver.log", O\_CREAT| O\_WRONLY | O\_APPEND,0644)) >= 0) {

    (void)write(fd,logbuffer,strlen(logbuffer));

    (void)write(fd,"\n",1);

    (void)close(fd);

  }

  gettimeofday(&t2,NULL);

  timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

  /\*printf("The time of loggbuffer sending to webserver.log：%lf\n",timeuse);\*/

}

void web(int fd, int hit)

{

  struct timeval t1,t2;

  double timeuse;

  int j, file\_fd, buflen;

  long i, ret, len;

  char \* fstr;

  static char buffer[BUFSIZE+1]; /\* 设置静态缓冲区 \*/

  gettimeofday(&t1,NULL);

  ret =read(fd,buffer,BUFSIZE);   /\* 从连接通道中读取客户端的请求消息 \*/

  gettimeofday(&t2,NULL);

  timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

  /\*printf("Time spent on reading cilent request from socket:%lf\n",timeuse); \*/

  if(ret == 0 || ret == -1) {  //如果读取客户端消息失败，则向客户端发送HTTP失败响应信息

    logger(FORBIDDEN,"failed to read browser request","",fd);

  }

  if(ret > 0 && ret < BUFSIZE)  /\* 设置有效字符串，即将字符串尾部表示为0 \*/

    buffer[ret]=0;

  else buffer[0]=0;

  for(i=0;i<ret;i++)  /\* 移除消息字符串中的“CF”和“LF”字符\*/

    if(buffer[i] == '\r' || buffer[i] == '\n')

      buffer[i]='\*';

  logger(LOG,"request",buffer,hit);

 /\*判断客户端HTTP请求消息是否为GET类型，如果不是则给出相应的响应消息\*/

  if( strncmp(buffer,"GET ",4) && strncmp(buffer,"get ",4) ) {

    logger(FORBIDDEN,"Only simple GET operation supported",buffer,fd);

  }

  for(i=4;i<BUFSIZE;i++) { /\* null terminate after the second space to ignore extra stuff \*/

    if(buffer[i] == ' ') { /\* string is "GET URL " +lots of other stuff \*/

      buffer[i] = 0;

      break;

    }

  }

  for(j=0;j<i-1;j++)   /\* 在消息中检测路径，不允许路径中出现“.” \*/

  {

    if(buffer[j] == '.' && buffer[j+1] == '.') {

      logger(FORBIDDEN,"Parent directory (..) path names not supported",buffer,fd);

    }

  }

  if( !strncmp(&buffer[0],"GET /\0",6) || !strncmp(&buffer[0],"get /\0",6) )

   /\* 如果请求消息中没有包含有效的文件名，则使用默认的文件名index.html \*/

    (void)strcpy(buffer,"GET /index.html");

  /\* 根据预定义在extensions中的文件类型，检查请求的文件类型是否本服务器支持 \*/

  gettimeofday(&t1,NULL);

  buflen=strlen(buffer);

  fstr = (char \*)0;

  for(i=0;extensions[i].ext != 0;i++)

  {

    len = strlen(extensions[i].ext);

    if( !strncmp(&buffer[buflen-len], extensions[i].ext, len)) {

      fstr =extensions[i].filetype;

      break;

    }

  }

  if(fstr == 0) logger(FORBIDDEN,"file extension type not supported",buffer,fd);

  gettimeofday(&t2,NULL);

  timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

  /\*printf("Verifing that whether the type of requesting file support:%lf\n",timeuse); \*/

  if(( file\_fd = open(&buffer[5],O\_RDONLY)) == -1) {  /\* 打开指定的文件名\*/

    logger(NOTFOUND, "failed to open file",&buffer[5],fd);

  }

  logger(LOG,"SEND",&buffer[5],hit);

  len = (long)lseek(file\_fd, (off\_t)0, SEEK\_END); /\* 通过lseek 获取文件长度\*/

   (void)lseek(file\_fd, (off\_t)0, SEEK\_SET); /\* 将文件指针移到文件首位置\*/

  (void)sprintf(buffer,"HTTP/1.1  200  OK\nServer:  nweb/%d.0\nContent-Length:  %ld\nConnection: close\nContent-Type: %s\n\n", VERSION, len, fstr); /\* Header + a blank line \*/

  logger(LOG,"Header",buffer,hit);

  (void)write(fd,buffer,strlen(buffer));

  /\* 不停地从文件里读取文件内容，并通过socket通道向客户端返回文件内容\*/

  gettimeofday(&t1,NULL);

  while (  (ret = read(file\_fd, buffer, BUFSIZE)) > 0 ) {

    (void)write(fd,buffer,ret);

  }

  sleep(1);  /\* sleep的作用是防止消息未发出，已经将此socket通道关闭\*/

  close(fd);

  gettimeofday(&t2,NULL);

  timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

  /\*printf("Reading the content of files and return to cilent through socket:%lf\n",timeuse);\*/

}

int main(int argc, char \*\*argv)

{

  int count=0;

  sem\_t\* psem;

  struct timeval t1,t2;

  unsigned long timeuse;

  int i,j, port, listenfd, socketfd, hit;

  socklen\_t length;

  static struct sockaddr\_in cli\_addr; /\* static = initialised to zeros \*/

  static struct sockaddr\_in serv\_addr; /\* static = initialised to zeros \*/

  //创建信号量,初始信号量为 1

  if((psem=sem\_open(SEM\_NAME, O\_CREAT,0666, 1))==SEM\_FAILED){

    perror("create semaphore error");

    exit(1);

  }

  int shm\_fd; //创建共享内存对象

  if((shm\_fd=shm\_open(SHM\_NAME,O\_RDWR| O\_CREAT,0666)) < 0){

    perror("create shared memory object error");

    exit(1);

  }

  /\* 配置共享内存段大小\*/

  ftruncate(shm\_fd, sizeof(int)); //将共享内存对象映射到进程

  void \* memPtr = mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

  if(memPtr==MAP\_FAILED){

    perror("create mmap error"); exit(1);

  }

  //为此内存区域赋值

    /\*for (i = 0; i < NUM; ++i) {    //信号量实现的临界区

      sem\_wait(psem);

      printf("Child Process count value: %d\n", (\*(int \*) memPtr)++);

      sem\_post(psem);

    }\*/

  gettimeofday(&t1,NULL);

  if( argc < 3  || argc > 3 || !strcmp(argv[1], "-?") )

  {

    (void)printf("hint: nweb Port-Number Top-Directory\t\tversion %d\n\n"

    "\tnweb is a small and very safe mini web server\n"

    "\tnweb only servers out file/web pages with extensions named below\n"

    "\t and only from the named directory or its sub-directories.\n"

    "\tThere is no fancy features = safe and secure.\n\n"

    "\tExample:webserver 8181 /home/nwebdir &\n\n"

    "\tOnly Supports:", VERSION);

    gettimeofday(&t2,NULL);

    timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

    printf("Analysing the parameters of command:%lf\n",timeuse);

    for(i=0;extensions[i].ext != 0;i++)

      (void)printf(" %s",extensions[i].ext);

    (void)printf("\n\tNot Supported: URLs including \"..\", Java, Javascript, CGI\n"

    "\tNot Supported: directories / /etc /bin /lib /tmp /usr /dev /sbin \n"

    "\tNo warranty given or implied\n\tNigel Griffiths nag@uk.ibm.com\n"  );

    exit(0);

  }

  if( !strncmp(argv[2],"/"   ,2 ) || !strncmp(argv[2],"/etc", 5 ) ||

    !strncmp(argv[2],"/bin",5 ) || !strncmp(argv[2],"/lib", 5 ) ||

    !strncmp(argv[2],"/tmp",5 ) || !strncmp(argv[2],"/usr", 5 ) ||

    !strncmp(argv[2],"/dev",5 ) || !strncmp(argv[2],"/sbin",6) )

  {

    (void)printf("ERROR: Bad top directory %s, see nweb -?\n",argv[2]);

    exit(3);

  }

  if(chdir(argv[2]) == -1){

    (void)printf("ERROR: Can't Change to directory %s\n",argv[2]);

    exit(4);

  }

  /\* 建立服务端侦听socket\*/

  gettimeofday(&t1,NULL);

  if((listenfd = socket(AF\_INET, SOCK\_STREAM,0)) <0)

    logger(ERROR, "system call","socket",0);

  port = atoi(argv[1]);

  if(port < 0 || port >60000)

    logger(ERROR,"Invalid port number (try 1->60000)",argv[1],0);

  serv\_addr.sin\_family = AF\_INET;

  serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

  serv\_addr.sin\_port = htons(port);

  if(bind(listenfd, (struct sockaddr \*)&serv\_addr,sizeof(serv\_addr)) <0){

    logger(ERROR,"system call","bind",0);

    printf("hahahaha\n");

  }

  if( listen(listenfd,64) <0)

    logger(ERROR,"system call","listen",0);

  gettimeofday(&t2,NULL);

  timeuse = (t2.tv\_sec-t1.tv\_sec)\*1000000.0+t2.tv\_usec-t1.tv\_usec;

  /\*printf("Time of establishing the webserver supervising socket:%lf\n",timeuse);\*/

  \* (unsigned long \*) memPtr= 0; //创建子进程

  for(hit=1; ;hit++) {

    length = sizeof(cli\_addr);

    if((socketfd = accept(listenfd, (struct sockaddr \*)&cli\_addr, &length)) < 0)

      logger(ERROR,"system call","accept",0);

    pid\_t pid=fork();

    struct timeval ts,te;

    unsigned long timespent;

    if (pid==0) //child process

    {

      gettimeofday(&ts,NULL);

      web(socketfd,hit);

      /\*close(listenfd);\*/

      gettimeofday(&te,NULL);

      printf("current pid is:%d\n",getpid());

      sem\_wait(psem);    //进入临界区

      timeuse = (te.tv\_sec-ts.tv\_sec)\*1000000+te.tv\_usec-ts.tv\_usec;

      \*(unsigned long \*)memPtr += timeuse;

      printf("Time spent on child%d process:%d us,total time is %d us \n",hit,timeuse,\*(unsigned long \*)memPtr);

      sem\_post(psem);    //释放资源

      exit(0);

    }

    else if (pid > 0) // parent process

    {

      (void)close(socketfd); //父进程关闭接收

    }

    else{

      perror("create childProcess error");

      exit(1);

    }

  }

  /\*if(munmap(memPtr,sizeof(int))==-1)

  {

    perror("unmap failed");

    exit(1);

  }

  if(close(shm\_fd)==-1)

  {

    perror("close shm failed");

    exit(1);

  }

  if(shm\_unlink(SHM\_NAME)==-1)

  {

    perror("shm\_unlink error");

    exit(1);

  }

  if(sem\_close(psem)==-1)

  {

    perror("close sem error");

    exit(1);

  }

  if(sem\_unlink(SEM\_NAME)==-1)

  {

    perror("sem\_unlink error");

    exit(1);

  }\*/

  exit(0);

}