Problem Set 4 :MMAP TEST

Scott Jin

2017-12-01

Contents

1 Code Listings **1**

2 Experimental Screenshots **13**

3 Narrative **15**

3.1 4-5 15

3.2 6 15

List of Figures

1 Test result for PS1-3 13

2 Test result for PS4-6 14

1 Code Listings

Scott Jin—p1.c

```c
/*
 * p1.c
 *
 *  Created on: Dec 2, 2017
 *      Author: scott
 */
#include <inttypes.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>

#define FILENAME "testfile"

int createFile(size_t length) {
  char c = 'A';
 int fd = open(FILENAME, O_RDWR|O_CREAT|O_TRUNC, 0666);
 if(fd==-1){
     fprintf(stderr,"Failed␣to␣open␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
   exit(EXIT_FAILURE);
 }
  for(int i=0; i < length; i++) {
    if(write(fd, &c, 1)!=1){
     fprintf(stderr,"Failed␣to␣write␣'A'␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
     exit(EXIT_FAILURE);
    }
   }
 if(lseek(fd, 0, SEEK_SET)==-1){
  fprintf(stderr,"Failed␣to␣lseek␣back␣to␣the␣start␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
  exit(EXIT_FAILURE);
   }
  return fd;
}
void SEGVHandler(int signo) {
  fprintf(stderr,"Caught␣SIGSEGV␣on␣writing␣to␣read␣only␣mmap:␣%s\n", strsignal(signo));
  exit(signo);
}
void BUSHandler(int signo) {
  fprintf(stderr,"Caught␣SIGBUS␣on␣writing␣to␣read␣only␣mmap<access␣to␣buffer␣that␣does␣not␣correspond␣to␣the␣file>:␣%s\n", strsignal(signo));
  exit(signo);
}
size_t GetFileSize(const char* filename) {
 struct stat fileStat;
 if (stat(filename, &fileStat) != 0) {
    fprintf(stderr,"Unable␣to␣get␣size␣of␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
 } else {
  fprintf(stderr,"TestFile[%s]␣size␣is:␣%jd\n",FILENAME, (intmax_t)fileStat.st_size);
  return  fileStat.st_size;
 }
}
int main(int argc, char ** argv ){
  off_t fileSize;
    size_t length=8192;
  char value_there;
  if(signal(SIGSEGV, SEGVHandler) == SIG_ERR){
    fprintf(stderr,"ERROR:occured while setting signal handler for signal SIGSEGV: %s\n",
        strerror(errno));
    exit(EXIT_FAILURE);
  }
  if(signal(SIGBUS, BUSHandler) == SIG_ERR){/*should not happen*/
    fprintf(stderr,"ERROR:occured while setting signal handler for signal SIGBUS: %s\n",
        strerror(errno));
    exit(EXIT_FAILURE);
  }
  int fd=createFile(length);
  fileSize=GetFileSize(FILENAME);
    fprintf(stderr,"Creating mapped_area[PROT_READ] with size : %jd\n",(intmax_t)(fileSize))
        ;
  char * mapped_area = mmap(NULL,(size_t)fileSize, PROT_READ, MAP_SHARED, fd, 0);
  if(mapped_area==MAP_FAILED){
     fprintf(stderr,"Failed to mmap testfile[%s]: %s\n",FILENAME,strerror(errno));
     exit(EXIT_FAILURE);
  }
    fprintf(stderr,"testing readibility for mapped_area\n");
    value_there=mapped_area[3];
    fprintf(stderr,"reading value from mapped_area:mapped_area[3]==%c\n",value_there);
    fprintf(stderr,"testing readibility for mapped_area:write a 1\n");
  mapped_area[3] = 1; /* SIGSEGV */
    if(mapped_area[3]==value_there) return 255;
  if(munmap(mapped_area,(size_t) fileSize)==-1){
    fprintf(stderr,"Failed to munmap testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  if(close(fd)==-1){
    fprintf(stderr,"Failed to close testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
    return EXIT_SUCCESS;
}
```

```
/*
 * p2.c
 *
 *  Created on: Dec 2, 2017
 *      Author: scott
 */
#include <inttypes.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FILENAME "testfile"

size_t GetFileSize(const char* filename) {
  struct stat fileStat;
  if (stat(filename, &fileStat) != 0) {
        fprintf(stderr,"Unable␣to␣get␣size␣of␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
        exit(EXIT_FAILURE);
  } else {
    fprintf(stderr,"TestFile[%s]␣size␣is:␣%jd\n",FILENAME, (intmax_t)fileStat.st_size);
    return  fileStat.st_size;
  }
}
int createFile(size_t length) {
    char c = 'A';
  int fd = open(FILENAME, O_RDWR|O_CREAT|O_TRUNC, 0666);
  if(fd==-1){
        fprintf(stderr,"Failed␣to␣open␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
        exit(EXIT_FAILURE);
  }
    for(int i=0; i < length; i++) {
        if(write(fd, &c, 1)!=1){
            fprintf(stderr,"Failed␣to␣write␣'A'␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME
                ,strerror(errno));
            exit(EXIT_FAILURE);
        }
      }
      if(lseek(fd, 0, SEEK_SET)==-1){
    fprintf(stderr,"Failed␣to␣lseek␣back␣to␣the␣start␣of␣filesize␣for␣testfile[%s]:␣%s\n",
        FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
      }
      return fd;
}
int main(int argc, char ** argv ){
  int flag;
  off_t fileSize;
  size_t length=8192;
    int buffSize;
  int fd=createFile(length);
  fileSize=GetFileSize(FILENAME);
    fprintf(stderr,"Creating␣mapped_area[PROT_WRITE]␣with␣size␣:␣%jd\n",(intmax_t)(fileSize)
        );
  char * mapped_area = mmap(NULL,(size_t)fileSize, PROT_WRITE, MAP_SHARED, fd, 0);
  if(mapped_area==MAP_FAILED){
     fprintf(stderr,"Failed␣to␣mmap␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
     exit(EXIT_FAILURE);
  }
  char Strtowrite[] = "This␣Assignment␣is␣real␣fun,␣dont␣let␣c␣bother␣us.";
  buffSize= sizeof Strtowrite;
  for (int i = 0; i < buffSize ; i++)
    mapped_area[i] = Strtowrite[i];

  if(munmap(mapped_area,(size_t) fileSize)==-1){
    fprintf(stderr,"Failed to munmap testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  char buff[buffSize]; /*declared here since string length shall not be changed */
  if(read(fd, buff, buffSize)==-1){
    fprintf(stderr,"Failed to read testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  if (!strcmp(buff,Strtowrite)) {
    flag=0;
    fprintf(stderr,"Update to an mmapped file with MAP_SHARED is visible to read(2).\n");
  } else {
    flag=1;
    fprintf(stderr,"Update to an mmapped file with MAP_SHARED is not visible to read(2).\n")
        ;
  }
  if(close(fd)==-1){
    fprintf(stderr,"Failed to close testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  exit(flag);
  return 0;
}
```

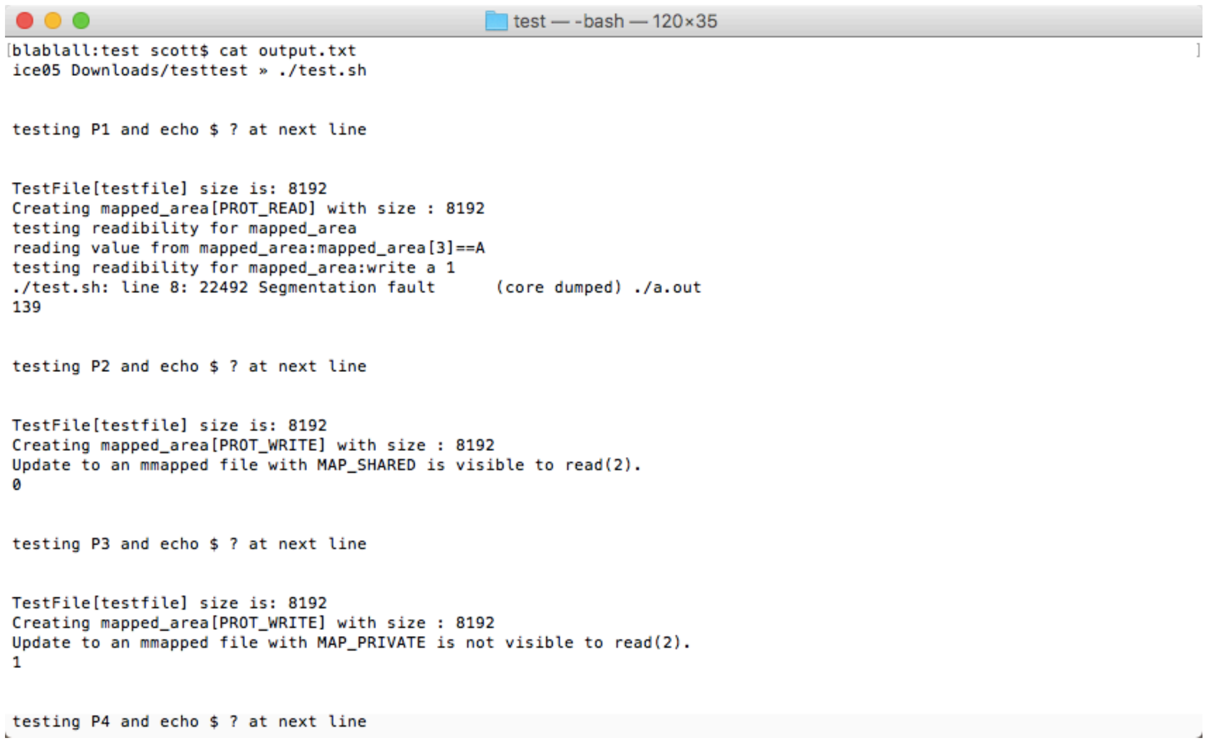
```
/*
 * p3.c
 *
 *  Created on: Dec 2, 2017
 *      Author: scott
 */
#include <inttypes.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FILENAME "testfile"

int createFile(size_t length) {
  char c = 'A';
  int fd = open(FILENAME, O_RDWR|O_CREAT|O_TRUNC, 0666);
  if(fd==-1){
        fprintf(stderr,"Failed␣to␣open␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
        exit(EXIT_FAILURE);
  }
  for(int i=0; i < length; i++) {
        if(write(fd, &c, 1)!=1){
            fprintf(stderr,"Failed␣to␣write␣'A'␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME
                ,strerror(errno));
            exit(EXIT_FAILURE);
        }
      }
  if(lseek(fd, 0, SEEK_SET)==-1){
    fprintf(stderr,"Failed␣to␣lseek␣back␣to␣the␣start␣of␣filesize␣for␣testfile[%s]:␣%s\n",
        FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
      }
  return fd;
}
size_t GetFileSize(const char* filename) {
  struct stat fileStat;
  if (stat(filename, &fileStat) != 0) {
        fprintf(stderr,"Unable␣to␣get␣size␣of␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
        exit(EXIT_FAILURE);
  } else {
    fprintf(stderr,"TestFile[%s]␣size␣is:␣%jd\n",FILENAME, (intmax_t)fileStat.st_size);
    return  fileStat.st_size;
  }
}
int main(int argc, char ** argv ){
  int flag;
  off_t fileSize;
  size_t length=8192;
    int buffSize;
  int fd=createFile(length);
  fileSize=GetFileSize(FILENAME);
  fprintf(stderr,"Creating␣mapped_area[PROT_WRITE]␣with␣size␣:␣%jd\n",(intmax_t)(fileSize));
  char * mapped_area = mmap(NULL,(size_t)fileSize, PROT_WRITE, MAP_PRIVATE, fd, 0);
  if(mapped_area==MAP_FAILED){
     fprintf(stderr,"Failed␣to␣mmap␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
     exit(EXIT_FAILURE);
  }
  char Strtowrite[] = "This␣Assignment␣is␣real␣fun.␣dont␣let␣b␣bother␣us";
  buffSize= sizeof Strtowrite;

  for (int i = 0; i < buffSize ; i++)
    mapped_area[i] = Strtowrite[i];

  if(munmap(mapped_area,(size_t) fileSize)==-1){
    fprintf(stderr,"Failed to munmap testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  char buff[buffSize]; /*declared here since string length shall not be changed */
  if(read(fd, buff, buffSize)==-1){
    fprintf(stderr,"Failed to read testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  if (!strcmp(buff,Strtowrite)) {
    flag=0;
    fprintf(stderr,"Update to an mmapped file with MAP_PRIVATE is visible to read(2).\n");
  } else {
    flag=1;
    fprintf(stderr,"Update to an mmapped file with MAP_PRIVATE is not visible to read(2).\n"
        );
  }
  if(close(fd)==-1){
    fprintf(stderr,"Failed to close testfile[%s]: %s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  exit(flag);
  return 0;
}
```

```
/*
 * p4.c
 *
 *  Created on: Dec 3, 2017
 *      Author: scott
 */
#include <inttypes.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FILENAME "something_in_the_middle"

int createFile(size_t length) {
  char c = 'A';
  int fd = open(FILENAME, O_RDWR|O_CREAT|O_TRUNC, 0666);
  if(fd==-1){
        fprintf(stderr,"Failed␣to␣open␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
      for(int i=0; i < length; i++) {
        if(write(fd, &c, 1)!=1){
          fprintf(stderr,"Failed␣to␣write␣'A'␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,
              strerror(errno));
          exit(EXIT_FAILURE);
        }
      }
      if(lseek(fd, 0, SEEK_SET)==-1){
    fprintf(stderr,"Failed␣to␣lseek␣back␣to␣the␣start␣of␣filesize␣for␣testfile[%s]:␣%s\n",
        FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
      }
      return fd;
}
size_t GetFileSize(const char* filename) {
  struct stat fileStat;
  if (stat(filename, &fileStat) != 0) {
        fprintf(stderr,"Unable␣to␣get␣size␣of␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
        exit(EXIT_FAILURE);
  } else {
    fprintf(stderr,"TestFile[%s]␣size␣is:␣%jd\n",FILENAME, (intmax_t)fileStat.st_size);
    return  fileStat.st_size;
  }
}
int main(int argc, char ** argv ){
  int page_size = getpagesize (  );
  fprintf(stderr,"Page␣size␣is:␣%d\n",page_size);
  int flag;
  off_t fileSize;
    off_t newfileSize;
    size_t length=8195;
  int fd=createFile(length);
  fileSize=GetFileSize(FILENAME);
    fprintf(stderr,"Creating␣mapped_area[PROT_WRITE␣|␣PROT_READ]␣with␣size␣:␣%jd\n",(
        intmax_t)(fileSize));
  char * mapped_area = mmap(NULL,(size_t)fileSize, PROT_WRITE | PROT_READ, MAP_SHARED, fd,
      0);
  if(mapped_area==MAP_FAILED){
     fprintf(stderr,"Failed␣to␣mmap␣testfil[%s]:␣%s\n",FILENAME,strerror(errno));
     exit(EXIT_FAILURE);
  }
  mapped_area[fileSize] = 'e';
  if(munmap(mapped_area,(size_t) fileSize)==-1){
    fprintf(stderr,"Failed␣to␣munmap␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  newfileSize=GetFileSize(FILENAME);
  if (newfileSize==fileSize) {
    flag=1;
    fprintf(stderr,"When␣a␣write␣is␣made␣one␣byte␣beyond␣the␣size␣of␣an␣mmapped␣file,␣for␣a␣
        file␣with␣a␣size␣that␣is␣not␣a␣multiple␣of␣of␣the␣page␣size,␣its␣size␣as␣reported␣by
        ␣stat(2)␣does␣not␣change.\n");
  } else {
    flag=0;
    fprintf(stderr,"When␣a␣write␣is␣made␣one␣byte␣beyond␣the␣size␣of␣an␣mmapped␣file,␣for␣a␣
        file␣with␣a␣size␣that␣is␣not␣a␣multiple␣of␣of␣the␣page␣size,␣its␣size␣as␣reported␣by
        ␣stat(2)␣changes.\n");
  }
  if(close(fd)==-1){
    fprintf(stderr,"Failed␣to␣close␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  exit(flag);
  return 0;
}
```

```c
/*
 * p5.c
 *
 *  Created on: Dec 3, 2017
 *      Author: scott
 */
#include <inttypes.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#define FILENAME "something_in_the_middle"

int createFile(size_t length) {
    char c = 'A';
  int fd = open(FILENAME, O_RDWR|O_CREAT|O_TRUNC, 0666);
  if(fd==-1){
          fprintf(stderr,"Failed␣to␣open␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
      exit(EXIT_FAILURE);
  }
    for(int i=0; i < length; i++) {
        if(write(fd, &c, 1)!=1){
            fprintf(stderr,"Failed␣to␣write␣'A'␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
            exit(EXIT_FAILURE);
        }
    }
    if(lseek(fd, 0, SEEK_SET)==-1){
        fprintf(stderr,"Failed␣to␣lseek␣back␣to␣the␣start␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
        exit(EXIT_FAILURE);
    }
    return fd;
}
size_t GetFileSize(const char* filename) {
  struct stat fileStat;
  if (stat(filename, &fileStat) != 0) {
          fprintf(stderr,"Unable␣to␣get␣size␣of␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
          exit(EXIT_FAILURE);
  } else {
    fprintf(stderr,"TestFile[%s]␣size␣is:␣%jd\n",FILENAME, (intmax_t)fileStat.st_size);
    return  fileStat.st_size;
  }
}
int main(int argc, char ** argv ){
  int page_size = getpagesize (  );
  fprintf(stderr,"Page␣size␣is:␣%d\n",page_size);
  int flag;
  off_t fileSize;
    off_t newfileSize;
    size_t length=8195;
    int fd=createFile(length);
  fileSize=GetFileSize(FILENAME);
    fprintf(stderr,"Creating␣mapped_area[PROT_WRITE␣|␣PROT_READ]␣with␣size␣:␣%jd\n",(intmax_t)(fileSize + 1));
  char * mapped_area = mmap(NULL,(size_t)fileSize, PROT_WRITE | PROT_READ, MAP_SHARED, fd, 0);
  if(mapped_area==MAP_FAILED){
      fprintf(stderr,"Failed␣to␣mmap␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
      exit(EXIT_FAILURE);
  }

    fprintf(stderr,"Writing␣'X'␣to␣offset:␣%jd\n",(intmax_t)(fileSize));
    mapped_area[fileSize] = 'X';

  if(lseek(fd, 16, SEEK_END)==-1){
      fprintf(stderr,"Failed␣to␣lseek␣past␣end␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
      exit(EXIT_FAILURE);
  }

  fprintf(stderr,"Writing␣'f'␣with␣write␣command␣to␣offset:␣%jd\n",(intmax_t)(fileSize + 16));
  unsigned char onemorebyte = 'f';
  if(write(fd, &onemorebyte, sizeof(unsigned char))==-1){
    fprintf(stderr,"Failed␣to␣write␣past␣end␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }

    fprintf(stderr,"lseeking␣back␣to␣end␣of␣the␣file␣at␣offset:%jd\n",(intmax_t)fileSize);
  if(lseek(fd, fileSize, SEEK_SET)==-1){
        fprintf(stderr,"Failed␣to␣lseek␣at␣middle␣of␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  };

    fprintf(stderr,"Reading␣1␣byte␣from␣the␣file[%s]␣with␣offset[%jd]\n",FILENAME,(intmax_t)fileSize);
  char buff[1];
  if(read(fd, buff, sizeof buff)==-1){
    fprintf(stderr,"Failed␣to␣read␣at␣middle␣of␣for␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }

    newfileSize=GetFileSize(FILENAME);

  if(buff[0]==0){
    fprintf(stderr,"read␣system␣call␣return=0\n");
  }else{
    fprintf(stderr,"read␣system␣call␣return=%c\n",buff[0]);
  }
  if (buff[0] == 'X') { /*weird BSD result?*/
      flag=0;
      fprintf(stderr,"If␣we␣create␣a␣\"hole\"␣in␣a␣file,␣any␣changes␣previously␣made␣in␣an␣mmapped␣region␣beyond␣the␣end␣of␣the␣file␣will␣be␣visible.\n");
    } else {
      flag=1;
      fprintf(stderr,"If␣we␣create␣a␣\"hole\"␣in␣a␣file,␣any␣changes␣previously␣made␣in␣an␣mmapped␣region␣beyond␣the␣end␣of␣the␣file␣will␣not␣be␣visible.\n");
    }
  if(munmap(mapped_area,(size_t) fileSize)==-1){
    fprintf(stderr,"Failed␣to␣munmap␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  if(close(fd)==-1){
    fprintf(stderr,"Failed␣to␣close␣testfile[%s]:%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  exit(flag);
  return 0;
}
```

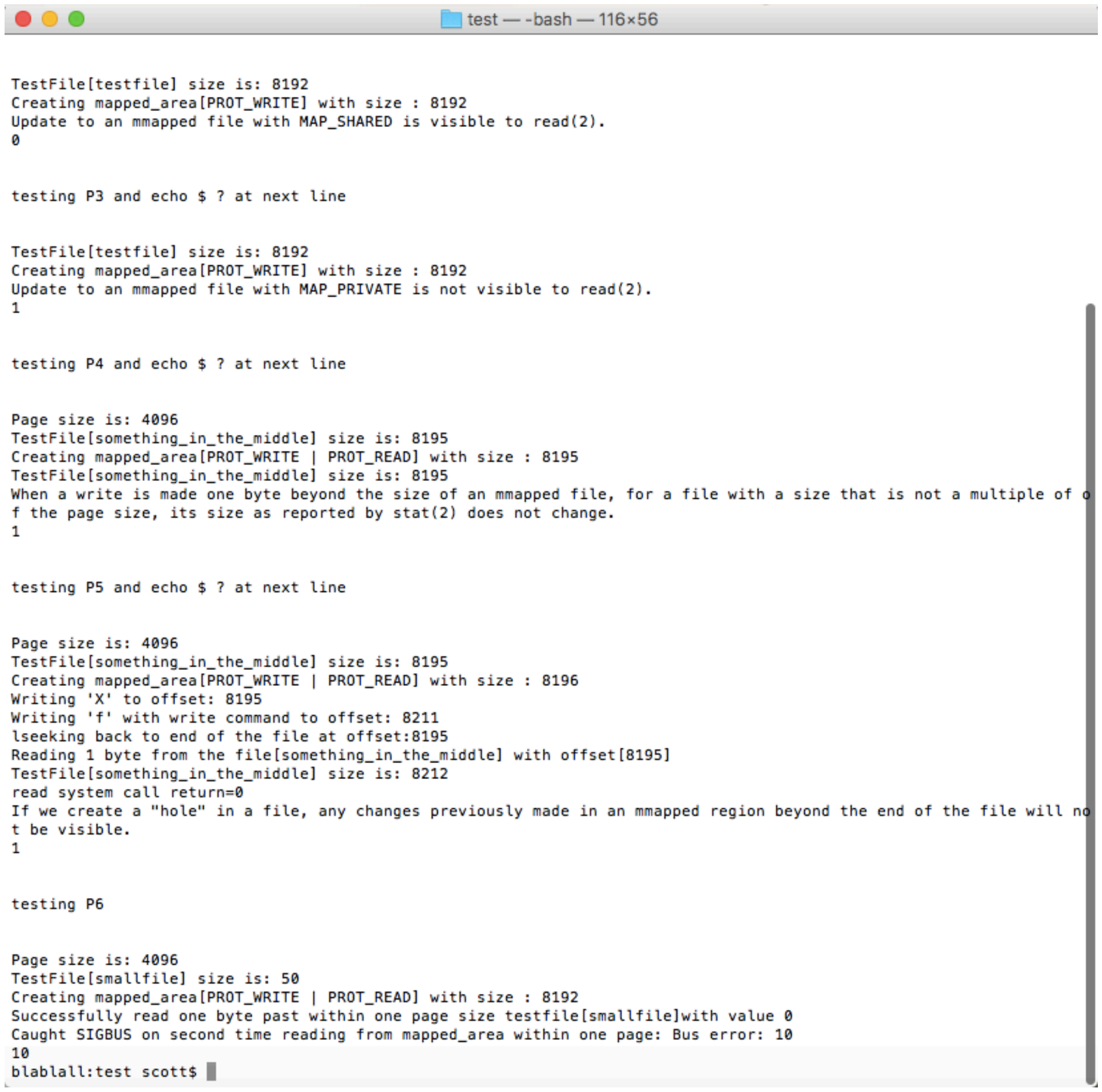
```c
/*  p6.c
 *  Created on: Dec 3, 2017
 *  Author: scott
 */
#include <inttypes.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FILENAME "smallfile"

char* errtime="first";

int createFile(size_t length) {
   char c = 'A';
 int fd = open(FILENAME, O_RDWR|O_CREAT|O_TRUNC, 0666);
 if(fd==-1){
     fprintf(stderr,"Failed␣to␣open␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
   exit(EXIT_FAILURE);
 }
  for(int i=0; i < length; i++) {
    if(write(fd, &c, 1)!=1){
      fprintf(stderr,"Failed␣to␣write␣'A'␣of␣filesize␣for␣testfile[%s]:␣%s\n",FILENAME
        ,strerror(errno));
      exit(EXIT_FAILURE);
    }
  }
  if(lseek(fd, 0, SEEK_SET)==-1){
  fprintf(stderr,"Failed␣to␣lseek␣back␣to␣the␣start␣of␣filesize␣for␣testfile[%s]:␣%s\n",
    FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
  }
  return fd;
}
size_t GetFileSize(const char* filename) {
 struct stat fileStat;
 if (stat(filename, &fileStat) != 0) {
     fprintf(stderr,"Unable␣to␣get␣size␣of␣testfile[%s]:␣%s\n",FILENAME,strerror(errno)
       );
     exit(EXIT_FAILURE);
 } else {
  fprintf(stderr,"TestFile[%s]␣size␣is:␣%jd\n",FILENAME, (intmax_t)fileStat.st_size);
  return  fileStat.st_size;
 }
}
void SEGVHandler(int signo) {
  fprintf(stderr,"Caught␣SIGSEGV␣on␣%s␣time␣reading␣from␣mapped_area␣within␣one␣page:␣%s\n
    ", errtime, strsignal(signo));
  exit(signo);
}
void BUSHandler(int signo) {
 fprintf(stderr,"Caught␣SIGBUS␣on␣%s␣time␣reading␣from␣mapped_area␣within␣one␣page:␣%s\n",
   errtime, strsignal(signo));
 exit(signo);
}
int main(int argc, char ** argv ){
 if(signal(SIGSEGV, SEGVHandler) == SIG_ERR){
  fprintf(stderr,"ERROR:occured␣while␣setting␣signal␣handler␣for␣signal␣SIGSEGV:␣%s\n",
    strerror(errno));
  exit(EXIT_FAILURE);
 }
 if(signal(SIGBUS, BUSHandler) == SIG_ERR){
  fprintf(stderr,"ERROR:occured␣while␣setting␣signal␣handler␣for␣signal␣SIGBUS:␣%s\n",
    strerror(errno));
  exit(EXIT_FAILURE);
 }
 int page_size = getpagesize (  );
 fprintf(stderr,"Page␣size␣is:␣%d\n",page_size);
 int flag;
 off_t fileSize;
 size_t length=50;
 int fd=createFile(length);
 fileSize=GetFileSize(FILENAME);
  fprintf(stderr,"Creating␣mapped_area[PROT_WRITE␣|␣PROT_READ]␣with␣size␣:␣%jd\n",(
    intmax_t)(2*page_size));
 char * mapped_area = mmap(NULL,(size_t)(2*page_size), PROT_WRITE | PROT_READ, MAP_SHARED,
   fd, 0);
 if(mapped_area==MAP_FAILED){
    fprintf(stderr,"Failed␣to␣mmap␣testfile[%s]:␣%s\n",FILENAME,strerror(errno));
    exit(EXIT_FAILURE);
 }
 char onebyte=mapped_area[fileSize+1];
 errtime="second"; /*For debug usage*/
 fprintf(stderr,"Successfully␣read␣one␣byte␣past␣within␣one␣page␣size␣testfile[%s]",
   FILENAME);
 if(onebyte==0){fprintf(stderr,"with␣value␣0\n");
 }else{  fprintf(stderr,"with␣value␣:%c\n",onebyte);}
 char onemorebyte=mapped_area[page_size+1];
 fprintf(stderr,"Successfully␣read␣one␣byte␣past␣one␣page␣size␣testfile[%s]␣with␣value=%c\n
   ",FILENAME,onebyte);
 exit(EXIT_SUCCESS) ;
}
```

2 Experimental Screenshots



```
blablall:test scott$ cat output.txt
ice05 Downloads/testtest » ./test.sh


testing P1 and echo $ ? at next line


TestFile[testfile] size is: 8192
Creating mapped_area[PROT_READ] with size : 8192
testing readibility for mapped_area
reading value from mapped_area:mapped_area[3]==A
testing readibility for mapped_area:write a 1
./test.sh: line 8: 22492 Segmentation fault      (core dumped) ./a.out
139


testing P2 and echo $ ? at next line


TestFile[testfile] size is: 8192
Creating mapped_area[PROT_WRITE] with size : 8192
Update to an mmapped file with MAP_SHARED is visible to read(2).
0


testing P3 and echo $ ? at next line


TestFile[testfile] size is: 8192
Creating mapped_area[PROT_WRITE] with size : 8192
Update to an mmapped file with MAP_PRIVATE is not visible to read(2).
1


testing P4 and echo $ ? at next line
```

Figure 1: Test result for PS1-3


test — -bash — 116×56

```
TestFile[testfile] size is: 8192
Creating mapped_area[PROT_WRITE] with size : 8192
Update to an mmapped file with MAP_SHARED is visible to read(2).
0


testing P3 and echo $ ? at next line


TestFile[testfile] size is: 8192
Creating mapped_area[PROT_WRITE] with size : 8192
Update to an mmapped file with MAP_PRIVATE is not visible to read(2).
1


testing P4 and echo $ ? at next line


Page size is: 4096
TestFile[something_in_the_middle] size is: 8195
Creating mapped_area[PROT_READ] with size : 8195
TestFile[something_in_the_middle] size is: 8195
When a write is made one byte beyond the size of an mmapped file, for a file with a size that is not a multiple of o
f the page size, its size as reported by stat(2) does not change.
1


testing P5 and echo $ ? at next line


Page size is: 4096
TestFile[something_in_the_middle] size is: 8195
Creating mapped_area[PROT_WRITE | PROT_READ] with size : 8196
Writing 'X' to offset: 8195
Writing 'f' with write command to offset: 8211
lseeking back to end of the file at offset:8195
Reading 1 byte from the file[something_in_the_middle] with offset[8195]
TestFile[something_in_the_middle] size is: 8212
read system call return=0
If we create a "hole" in a file, any changes previously made in an mmapped region beyond the end of the file will no
t be visible.
1


testing P6


Page size is: 4096
TestFile[smallfile] size is: 50
Creating mapped_area[PROT_WRITE | PROT_READ] with size : 8192
Successfully read one byte past within one page size testfile[smallfile]with value 0
Caught SIGBUS on second time reading from mapped_area within one page: Bus error: 10
10
blablall:test scott$ 
```

Figure 2: Test result for PS4-6

3 Narrative

3.1 4-5

The outputs from problem 4 was expected[invisible] with the stat system call. When file [something in the middle] was created, the kernel allocated two pages for it because its size is larger than one but not two page sizes. The page size is larger than the file size, so the remainder of the page is filled with zeros. when a write is made outside of the file as I did in Problem 4, the zero I am writing to was replaced whatever I wrote with. And the file size shall not change by stat(2) system call since this action has no effect onthe size of the file. The outputs from problem 5 was unexpected[invisible] with the read system call. When the file "something in the middle" was created, same procedures are made by kernel as Problem 5. Then I lseek beyond the file size and write to it, the kernel should update the header in the file and ignore what is in the hole I just created, since the page remainder was filled with zeros and there is no meaning for extending file size to fill the file with zeros again.

According to "man 2 lseek" :the following quote was acquired:

"The lseek() function allows the file offset to be set beyond the end of the file (but this does not change the size of the file).If data is later written at this point, subsequent reads of the data in the gap (a "hole") return null bytes until data is actually written into the gap."

Maybe the read system was contrived to follow the lseek description and therefore actually go back to the original file size and fill in zeros until the new data offset and write the data I was actually writing. Note MAP SHARED was used to create map all along so the write to the page was reflected in the memory directly by other process or Kernel.

3.2 6

As explained in the lecture notes The mapped region extended beyond the current end of the file, made by declaring the len parameter larger than the file size [filesize+1]. If the process attempts to access memory which corresponds to just beyond the current end of file, but not beyond a page size boundary, it will see bytes with a value of 0, since all the remainder of the file was filled with zeros. However, when the process tries to go to the second page, even though the memory address is within the bounds of the mapped region, the kernel will be unable to satisfy the page fault, because the backing store does not exist in the filesystem since the page is not used and the kernerl did not allocate it in the Page Global Directory. Under this condition, the kernel will deliver a SIGBUS[Attempted access to a portion of the buffer that does not correspond to the file (for example, beyond the end of the file, including the case where another process has truncated the file).] to the process. The Experimental results prove this.