

פתרון מבחן במערכות הפעלה

ערן טרומר, מועד א' סמסטר א' תשע"ב

19 בפברואר 2012

(בגופן מוטה: הבהרות שנכתבו על הלוח)

הוראות

משך הבחינה שלוש שעות. לא תינתן הארכה.
יש לענות על כל השאלות. תשובה ללא נימוק לא תזכה באף נקודה.
יש לענות על כל השאלות בגוף הבחינה במקום המיועד לכך. המקום המיועד מספיק לתשובות מלאות ומדויקות, ולכן גם לבעלי כתב יד דחוס מומלץ לקצר בדבריהם.
יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תפסל.
תשובות במחברת הבחינה לא תבדקנה.
יש למלא מספר סידורי (מספר מחברת) ומספר ת"ז על כל דף של טופס הבחינה.
אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט.

בהצלחה!

שאלה 1 (19 נק')

א. מנו 3 דרכים שונות מהותית בהן תהליך משתמש יכול לגרום להרצה של קוד בגרעין מערכת ההפעלה באופן מידי, כלומר לכך שפקודת המכונה הבאה שתרוץ תהיה בקוד הגרעין. (6 נק')

1. קריאת מערכת ע"י פסיקת תוכנה (ובפרט גישה לקבצים, הקצאת זיכרון וירטואלי לתהליך, סיום ריצה ופעולות סינכרון).
2. ביצוע פקודת מכונה המותרת רק במצב מיוחס, או פקודת מכונה לא קיימת (כלומר PC מצביע לערך חסר פירוש כפקודת מכונה).
3. גרימת חריג דף (ובפרט, גישה לדף שאינו קיים בטבלת הדפים, או אינו בזיכרון, או גישת קריאה/כתיבה/הרצה לדף שרשומתו בטבלת הדפים אינו מרשה גישה כזו או אינה מרשה גישה במצב לא-מיוחס).
4. גרימת חריג אריתמטי (לדוגמה, חלוקה ב-0 של מספר נקודה-צפה).

טעויות נפוצות:

- אזכור של מספר פעולות אשר מתרגמות בעצם לקריאות מערכת דומות. לדוגמה, אין הבדל רלוונטי במנגנון בין גישה לקובץ עם הרשאות מתאימות לגישה לקובץ ללא הרשאות מתאימות; שתיהן קריאות מערכת דומות, ורק פעולת קוד הגרעין אח"כ שונה קצת.
- קריאות לפונקציית `malloc` יטופלו בתוך תהליך המשתמש ע"י מערכת ניהול ה-`heap` של ספריית ה-C, אלא אם כן במקרה נגמר הזיכרון הווירטואלי המוקצה וצריך לבקש עוד מהגרעין ע"י קריאת מערכת.
- מיפוי זיכרון וירטואלי לפיזי נעשה על ידי חומרה ולכן, אלא אם כן במקרה נגרם חריג, לא ירוץ קוד גרעין.

- סיום רגיל של תהליך נעשה גם הוא ע"י קריאת מערכת (exit), ולא באופן מסתורי אחר.

ב. בארכיטקטורות "פגומות" שאינן ניתנות לווירטואליזציה ישירה, ניתן לממש בכל זאת רב-פקח ע"י תרגום קוד בזמן ריצה: כל פעם שקטע קוד חדש רץ במכונה הווירטואלית, הוא "יתוקן" על ידי הרב-פקח כך שפקודות בעייתיות יוחלפו. כיצד יכול הרב-פקח לדעת ולהתערב כל פעם שקוד חדש רץ, בלי להאט ריצה של קוד אשר כבר תוקן? (אין צורך להסביר איך הקוד יתוקן.) (8 נק')

הרב-פקח יתערב באמצעות מנגנון פסיקות הדף. כל דף חדש במכונה הווירטואלית יסומן כבלתי ניתן להרצה, בטבלת הדפים האמיתית (כלומר טבלת הצללים או הטבלה המארכת, בהתאם למימוש). לכן בפעם הראשונה שקוד ירוץ, הוא יגרום לפסיקת דף אשר תפעיל את הרב-פקח. אז הרב-פקח יתקן את הקוד בדף, ויסמן את הדף כניתן להרצה (אבל בלתי ניתן לכתיבה, כדי לתפוס שינויים לצורך תיקון מחדש).

הערות: פרטי המימוש תלויים בארכיטקטורה, וכל גרסה סבירה התקבלה. השאלה הוצגה בעל-פה בכיתה כשאלה למחשבה, ובהמשך אותו השיעור דנו במנגנון האנלוגי בעזרתו הרב-פקח יכול להתערב כאשר המכונה הווירטואלית כותבת לזיכרון, על ידי סימון דפים כבלתי ניתנים לכתיבה. הפתרון דומה גם למנגנון copy-on-write. תשובות המניחות שהרב-פקח מכיר את נבכי מערכת הקבצים ומערכת ההפעלה בתוך המכונה הווירטואלית אינן נכונות. התקבלו כנכונות תשובות אשר לא התייחסו לבעיה הספציפית של ארכיטקטורות פגומות כפי שנדונה בכיתה, אך תארו מנגנון חלופי סביר.

ג. המשך הסעיף הקודם: האם יש צורך לבדוק ולתקן כל קוד אשר רץ (בפעם הראשונה) במכונה הווירטואלית, או שישנם מקרים בהם הרב-פקח יכול לדעת בקלות שאין צורך בבחינת הקוד? נמקו. (5 נק')

כאשר המכונה הווירטואלית היא במצב לא-מיוחס (ווירטואלי), אין צורך לבחון את הקוד שהיא מריצה: המצב האמיתי והווירטואלי זהים ולכן גם פקודות מכונה "פגומות" מתנהגות כמצופה מבחינת האורחת. הרב-פקח יודע מתי המכונה הווירטואלית במצב מיוחס, שכן כל כניסה ויציאה עוברת דרכו.

טעות נפוצה: יש צורך לבדוק גם קוד שהרב-פקח "מכיר" או "יודע שבו זהה למערכת המארכת" (כיצד ידע זאת?!), מאחר שבארכיטקטורה פגומה, קוד זה עלול לרוץ לא נכון במצב מיוחס וירטואלי אבל לא-מיוחס פיזי.

שאלה 2 (36 נק')

מנהל המערכת של מחשב לינוקס כתב תוכנית בשם `post_announcement` אשר מקבלת כקלט מחרוזת, וכותבת אותה בסוף הקובץ `/shared/last_announcement` אשר הרשאותיו `.rwxr--r-- root root`. אף תוכנית אחרת לא כותבת לקובץ זה. כדי לאפשר לכל המשתמשים להפעיל את `post_announcement`, הוא נתן לה הרשאות `rwxr-xr-x root root` עם ביט `setuid` דולק.

א. האם אכן `/shared/last_announcement` יכיל את המחרוזת שניתנו כקלט ל-`post_announcement`, זו אחר זו? נמקו. (5 נ')

התוכנית מצליחה לכתוב, השאלה היא אם "זו אחר זו".

הכתיבה תתאפשר: כל משתמש יכול להריץ את `post_announcement`, ובגלל ה-`setuid` היא תרוץ תחת הרשאות `root`. אבל במימוש נאיבי של כתיבה לסוף הקובץ, כאשר מספר עותקים של `post_announcement` מורצים במקביל, כתיבה אחת עלולה לדרוס כתיבה אחרת, סדר הכתיבה לקובץ יכול להיות שונה מסדר ההרצה, וכו'. נדרש מנגנון סנכרון כדי למנוע זאת.

ב. מסתבר שהרשאות הספרייה `/shared` הן `rwxrwxrwx root root`. כיצד יכול משתמש רגיל במערכת לכתוב לכל קובץ שירצה בכל מקום במערכת הקבצים? (5 נק')

המשתמש יכול להחליף את הקובץ `/shared/last_announcement` במצביע סימבולי לכל מסלול שיחפוץ במערכת הקבצים, ואז להריץ את `post_announcement` כדי שיכתוב לקובץ זה (תחת הרשאות `root`, בזכות `setuid`).

הערה: זו התקפה אנלוגית ל-`fingerd`, שם היה מדובר בקריאה מקובץ שעלול להיות מצביע סימבולי.

טעויות נפוצות:

- באופן כללי, אין שום דבר מיוחד בספרייה הניתנת לכתיבה ע"י כולם (כל אחד יכול ליצור כזו), ולכן כל תשובה שמשתמשת רק בהרשאות של `/shared` וְלא נעזרת בתרחיש הספציפי עם תוכנית `setuid`, לא יכולה להיות נכונה.

- אין למשתמש דרך ישירה לשנות את התוכנית `post_announcement`, שכן אין לו הרשאות כתיבה לקובץ זה.

- בלינוקס הרשאות ספרייה לא נורשות על ידי הקבצים שבה, שהרי לכל קובץ יש הרשאות מפורטות משלו.

- יש ליצור `symbolic link` ולא `hard link`, שכן כדי ליצור `hard link` אל קובץ נדרשת גישה לספרייה המכילה אותו (חייבת להיות דרישה כזו, אחרת ניתן היה לעקוף מגבלת הרשאות של ספריות).

- גישה דרך מצביע סימבולי נעשית לפי הרשאות הקובץ המוצבע (אין אפשרות שפויה אחרת).

- לא ניתן לעשות `mount`, שכן פעולה זו דורשת כמובן הרשאות `root`.

ג. בהינתן היכולת לכתוב לכל קובץ: כיצד יכול משתמש, בעזרת שורת shell או תוכנית קטנטנה, לגרום לכך שבפעם הבאה שהמחשב יכובה ויודלק, המחשב יתקע מיד (עוד לפני טעינת מערכת ההפעלה)? (5 נק')

תיקון לשאלה: יש לתוכנית `post_announcement` אופציה נוספת, לכתוב הודעה החל מתחילת `./shared/last_announcement`.

תיקון חלופי שקול: "בהינתן היכולת לכתוב לתחילת כל קובץ..."

המשתמש ישבש את תהליך טעינת מערכת ההפעלה, על ידי כתיבת קוד משובש ל-MBR (על ידי כתיבה לקובץ ההתקן `/dev/sda`, או אחד מהחלקים האחרים של שרשרת הטעינה (לדוגמה `/boot/grub/*`), או מערכת הקבצים עצמה.

טעות נפוצה: את קוד ה-BIOS עצמו אי אפשר לשבש; הוא מאוכסן בשבב יעודי אשר תוכנו (לרוב) לא ניתן לשינוי ע"י מערכת ההפעלה. לכן צריך לפגוע בשלב מאוחר יותר של תהליך הטעינה.

ד. האם כדאי לאכסן את אזור דפדוף בזיכרון מסוג `SSD/flash`, במקום בדיסק קשיח? ציינו יתרון וחסרון. (6 נק')

יתרונות:

- גישה אקראית מהירה (לא צריך להמתין להזזת רכיב מכני), מתאימה לדפוס האקראי-למדי של גישה להתקן הדפדוף.
- קצב העברה גבוה לקריאה (וכן לכתובה רצופה), לפחות בסוגים מסוימים של SSD.

חסרונות:

- כתיבות קטנות (של דף בודד) הן איטיות, כי צריך לשכתב בלוקים שלמים.
- שחיקה בעת כתיבה. חמור במיוחד בהקשרנו, כי אל התקן דפדוף יש כתיבות רבות (והן אף "מנופחות" בגלל החיסרון הקודם).

טעויות נפוצה:

- SSD לא נשחק בקריאה, רק בכתיבה.
- היו שטענו ש-SSD יקר מאד, והיו שטענו להפך... אם נדייק, ל-SSD עלות שולית (ליחידת נפח אחסון) גבוהה אבל עלות בסיסית (של ה"ברזלים" והבקר) נמוכה משל דיסק קשיח. זיכרון מטמון אופייני דורש מעט מקום (GB בודדים, אותו סדר גודל כמו הזיכרון הראשי). לכן שיקול העלות תלוי מאד בנסיבות (לדוגמה האם יש במערכת ממילא דיסק קשיח), וזו אינו תשובה מוצלחת לשאלה.
- SSD אמנם לא נדיף, אבל כך גם דיסק קשיח, ובכל מקרה נדיפות אינה רלוונטית לזיכרון דפדוף (התוכן רלוונטי רק כל עוד המחשב עובד, כלומר יש אספקת חשמל).

ה. במערכת קבצים מסורתית מבוססת `inodes`, כמה גישות לדיסק נדרשות לכל

הפחות כדי לקרוא את הבית ה-8097 בקובץ `/home/moishe/mail`? הניחו שגודל בלוק הוא 1024 בתים, `inode` מכיל 6 מצביעים ישירים לבלוקים ועוד 3 מצביעים עקיפים (`direct, double indirect, triple indirect`), כל מדריך מאוכסן בבלוק יחיד, ובתחילת הפעולה נמצא בזכרון רק ה-`inode` של שורש מערכת הקבצים. נמקו. (5 נק')

סה"כ 8 גישות לקריאת בלוק, בשרשרת ההצבעות הבאה:
מדריך של השורש /,

inode של home, מדריך של home,

inode של moishe, מדריך של moishe,

inode של mail, בלוק מצביעי direct של mail, בלוק מידע של הקובץ.

טעות נפוצה: לכל ספרייה, כולל השורש, יש מדריך (רשימה של תוכן הספרייה, מאוכסנת באופן דומה לתוכן קובץ), וגם-inode (מצביעים לרשימה הנ"ל, הרשאות וכו'). הם יושבים במקומות שונים בדיסק.

ו. כאשר גולש מבצע חיפוש בגוגל, האם גוגל יודע מה כתובת ה-MAC של מחשבו? נמקו. (5 נק')

לא. כתובת ה-MAC נמצאת בתחילית של רמת המיקשר, אשר מוחלפת בכל צעד ניתוב. לכן כתובת ה-MAC של המשתמש נראית רק במקטע האתרנט המקומי שלו.

הבהרה: מנות ה-TCP/IP אשר שרת גוגל מקבל מהמשתמש אמנם נושאות את כתובת ה-IP של המשתמש (אם אין NAT), אבל כתובת ה-MAC שבהן היא זו של הנתב אליו מחובר שרת גוגל.

ז. בפרוטוקול TCP, האם כדאי לשלוח מנה כל פעם שהחלון מאפשר זאת? נמקו. (5 נק')

לא תמיד. לדוגמה, "סינדרום החלון האוילי": אם החלון פתוח אך קטן מאד, כנראה שהחוצץ בצד המקבל כמעט מלא; כדאי לחכות שהתהליך המקבל יקרא עוד מהמידע שכבר הגיע (וכך יגדיל את החלון), במקום לשלוח מנות קטנות עם תקורה גבוהה. ראינו מספר אופטימיזציות כאלה.

שאלה 3 (45 נק')

In this question, you are asked to complete a code for a remote logging systems. The system allows logging from several "slave" computers to a single file at a remote server. At each "slave" computer, there will be multiple "application" processes that will write log entries into a shared memory buffer ("log buffer"). Each log entry will consist of a single DWORD value. At each "slave" computer there will be also a single "LogTransmitter" process that will transmit the "log buffer" to a remote server via TCP connection. The "LogTransmitter" will send the data as soon as it will be completely filled by log entries. All "application" processes should wait until "LogTransmitter" will finish sending "LogBuffer" and only then, they will be able to continue to write log entries. At the remote server, the "LogReceiver" process receives the data and appends it to a log file.

General instructions:

- No need to check for error conditions
 - Follow provided code template. Don't add code lines or function parameters.
 - Use infinite timeouts for all wait functions, and assume that the "LogReceiver" and "LogTransmitter" processes run forever.
- a. [5 points] Complete the implementation of the function below that appends a chunk of data to a file.

```
void AppendToFile(char* pBuffer, DWORD dwSize, LPCTSTR fileName){
    DWORD dwWritten=0;
    HANDLE hFile=CreateFile(fileName,
                            GENERIC_READ|GENERIC_WRITE,
                            FILE_SHARE_READ,
                            NULL,OPEN_ALWAYS,0,NULL);
    SetFilePointer(hFile, 0, NULL, FILE_END);
    WriteFile(hFile,pBuffer,dwSize,&dwWritten,NULL);
    CloseHandle(hFile);
}
```

- b. [15 points] Each time the "LogTransmitter" needs to send the log buffer to the "LogReceiver" it will connect to the server, will send a buffer of **NUM_LOG_ENTRIES DWORD's** to the server and then will disconnect. Complete the code for the rest of "LogReceiver". You don't need to handle concurrent connections.

```

#define NUM_LOG_ENTRIES      5
#define PORT_NUM             2134
#define LOG_FILE_NAME       _T("c:\\logfile.log")

int _tmain(int argc, _TCHAR* argv[]){
    WSADATA wsaData;
    WSASStartup(MAKEWORD(2,2), &wsaData);
    RunLogReceiver(PORT_NUM,LOG_FILE_NAME,
                  NUM_LOG_ENTRIES*sizeof(DWORD));

    return 0;
}

void RunLogReceiver(WORD port, LPCTSTR fileName,DWORD dwMessageSize){
    char* buf=(char*)malloc(dwMessageSize);
    SOCKET hLS= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in service;
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr("127.0.0.1");
    service.sin_port = htons(port);
    bind(hLS, (SOCKADDR*)&service, sizeof(service));
    listen(hLS,1);
    do{
        sockaddr client_addr;
        int nLen=sizeof(client_addr);
        SOCKET hRS=accept(hLS,&client_addr,&nLen);
        int nLeft=dwMessageSize;
        int iPos=0;
        do {
            int n=recv(hRecvSocket,buf+iPos,nLeft,0);
            nLeft-=n;
            iPos+=n;
        }while(nLeft);
        closesocket(hRS);
        AppendToFile(buf,dwMessageSize,fileName);
    }while(1);
}

```



```

}

void RunLogTransmitter(LPCSTR ipAddr,WORD port_num){
    MMF_OBJ mmf;
    CreateMMFObject(&mmf);
    do{
        mmf.pSharedData->dwNumEntriesWritten=0;
        ReleaseSemaphore(mmf.hFreeEntries,NUM_LOG_ENTRIES,NULL);
        WaitForSingleObject(mmf.hReadyToSend,INFINITE);
        SendToServer(ipAddr,port_num,
            (char*)(mmf.pSharedData->buffer),
            sizeof(mmf.pSharedData->buffer));
    }while(1);
}

void WriteLogEntry(MMF_OBJ* pMMF, DWORD dwLogEntry){
    HANDLE hGuard=CreateMutex(NULL,FALSE,GUARD_NAME);
    WaitForSingleObject(pMMF->hFreeEntries,INFINITE);
    WaitForSingleObject(hGuard,INFINITE);
    SHARED_DATA* pSharedData=pMMF->pSharedData;
    pSharedData->buffer[pSharedData-
>dwNumEntriesWritten]=dwLogEntry;
    pSharedData->dwNumEntriesWritten++
    if(pSharedData->dwNumEntriesWritten==NUM_LOG_ENTRIES)
        SetEvent(pMMF->hReadyToSend);
    ReleaseMutex(hGuard);
    CloseHandle(hGuard);
}

void DestroyMMFObject(MMF_OBJ* pMMF){
    UnmapViewOfFile(pMMF->pSharedData);
    CloseHandle(pMMF->hReadyToSend);
    CloseHandle(pMMF->hFreeEntries);
    CloseHandle(pMMF->hMMF);
}

void SendToServer(LPCSTR ipAddr,WORD port_num, char* buf,DWORD
nBytesToSend){
    int iPos=0;
    sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ipAddr);
    server_addr.sin_port = htons(port_num);
    SOCKET hS= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    connect( hS, (SOCKADDR*)&server_addr, sizeof(server_addr));
    while(nBytesToSend){
        int n=send(hSocket,buf+iPos,nBytesToSend,0);
        nBytesToSend-=n;
        iPos+=n;
    }
    closesocket(hSocket);
}

```