

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 מחשבוני 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')

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')

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')

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` יכול את המחרוזות שניתנו כקלט ל-`post_announcement`, זו אחר זו? נמקו. (5 נ')
התוכנית מצליחה לכתוב, השאלה היא אם "זו אחר זו".

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

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..."*

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

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')

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

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

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){
    _____ dwWritten=0;
    _____ hFile=_____ (
        _____,
        _____,
        FILE_SHARE_READ,
        NULL, OPEN_ALWAYS,
        0, NULL);
    _____ ( _____, _____, NULL, FILE_END);
    _____ ( _____, _____,
        _____, _____, NULL);
    _____;
}
```

- 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);
    _____ hLS= _____(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 = _____;
    _____(_____,
                (SOCKADDR*) &_____, sizeof(_____) );
    _____(_____,1);
    do{
        sockaddr client_addr;
        int nLen=sizeof(client_addr);
        _____ hRS= _____(_____,
                                    &client_addr,&nLen);

        int nLeft=dwMessageSize;
        int iPos=0;
        do {
            int n= _____(
                _____,
                _____,
                _____,0);
            _____;
            _____;
        }while(_____) ;
        _____;
        AppendToFile(buf,dwMessageSize,fileName);
    }while(1);
}

```

- c. [25 points] The "LogTransmitter" and multiple "application" processes will use a common library (DLL) for accessing a logging buffer which is implemented as shared memory. Below are sample main functions for the "LogTransmitter" and "application" process.

```
int _tmain(int argc, _TCHAR* argv[]){
    WSADATA wsaData;
    WSASStartup(MAKEWORD(2,2), &wsaData);
    RunLogTranmitter("127.0.0.1",PORT_NUM);
    return 0;
}
int _tmain(int argc, _TCHAR* argv[]){
    MMF_OBJ mmf;
    CreateMMFObject(&mmf);
    for (int i=1;i<100;i++)
        WriteLogEntry(&mmf,rand());
    DestroyMMFObject(&mmf);
    return 0;
}
```

Complete code below. Make sure that you code will work for any possible process scheduling scenario.

```
#define NUM_LOG_ENTRIES          5
#define PORT_NUM                 2134
#define FREE_ENTRIES_NAME       _T("FreeEntries")
#define MMF_NAME                 _T("MMF")
#define READY_TO_SEND_NAME      _T("ReadyToSend")
#define GUARD_NAME              _T("Guard")

struct SHARED_DATA{
    _____ buffer[NUM_LOG_ENTRIES];
    _____ dwNumEntriesWritten;
};

struct MMF_OBJ{
    _____ hMMF;
    SHARED_DATA* pSharedData;
    HANDLE hFreeEntries;
    HANDLE hReadyToSend;
};

void CreateMMFObject(MMF_OBJ* pMMF){
    pMMF->hMMF= _____ ( _____ ,
    NULL, PAGE_READWRITE, 0,
    _____ ,
    _____ );
    pMMF->pSharedData=( _____ ) _____ (
    _____ ,
    FILE_MAP_ALL_ACCESS
```

```

        ,0,0,
        _____);
pMMF->hReadyToSend=_____ (NULL,
        _____,
        _____);
pMMF->hFreeEntries=_____ (NULL,
        0, NUM_LOG_ENTRIES, _____);
}

```

```

void RunLoggingTranmitter(LPCSTR ipAddr,WORD port_num){
    MMF_OBJ mmf;
    CreateMMFObject(&mmf);
    do{
        //tell applications to continue
        _____=0;
        _____(pMMF->hFreeEntries, _____, NULL);
        //wait till buffer is ready
        _____( mmf.hReadyToSend, _____);
        SendToServer(ipAddr, port_num,
            (char*)(mmf.pSharedData->buffer),
            sizeof(mmf.pSharedData->buffer));
    }while(1);
}

```

```

void WriteLogEntry(MMF_OBJ* pMMF, DWORD dwLogEntry){
    _____ hGuard=_____ (_____ ,
        _____ ,
        _____);
    _____;
    _____;
    SHARED_DATA* pSharedData=pMMF->pSharedData;
    _____=dwLogEntry;
    pSharedData->dwNumEntriesWritten++;
    if(pSharedData->dwNumEntriesWritten==NUM_LOG_ENTRIES){
        _____};
    _____;
    _____;
}

```

```
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);
}
```