ИТМО Кафедра Информатики и прикладной математики

Отчет по лабораторной работе №5 «Объекты синхронизации»

Выполнил: студент группы P3217
Плюхин Дмитрий
Преподаватель: Зыков А. Г.

2017 год

1. Задание

Исследование на конкретном примере следующих объектов синхронизации:

1. критические секции
2. мьютексы
3. семафоры
4. события

Задачу для синхронизации выбрать на свое усмотрение.
Задачи для каждого метода синхронизации должны быть различными. Задачи должны наглядно демонстрировать выбранный метод синхронизации и учитывать его особенности. Студент, сдающий работу должен аргументированно обосновать задачу, выбранную для синхронизации и метод синхронизации.

2. Листинг основной части программы

В качестве задачи для синхронизации была выбрана разработка простейшей многопользовательской игры. Далее приводится описание всех случаев использования каждого объекта синхронизации.

2.1 Мьютекс

Подзадачей использования мьютексов в данном случае является чтение и запись данных зарегистрировавшегося пользователя в «базу данных», представленную текстовым файлом, содержащим логины и пароли. Поскольку мьютекс является именованным объектом синхронизации, что обеспечивает возможность его использования в разных процессах, а также позволяет допустить до единовременной работы с разделяемым ресурсом только один поток исполнения, его использование в данном контексте является наиболее практичным решением задачи синхронизации. Далее приведены фрагменты кода, связанные непосредственно с использованием мьютекса.

```
HANDLE getFileMutex(){
  HANDLE hMutex = OpenMutex(SYNCHRONIZE, FALSE, "fileMutex");
  if (hMutex == NULL){
    hMutex = CreateMutex(NULL, FALSE, "fileMutex");
  }
  return hMutex;
}

int writeUser(HANDLE fileMutex, const char* fileName, string login, string password){
  DWORD dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);
  ofstream fout(fileName, ios_base::app);
  fout <<"$"<<login<<"/"<<password;
  fout.close();
  ReleaseMutex(fileMutex);
  return 0;
}

int admitUser(int championship){
  HANDLE fileMutex = getFileMutex();
  string login = getData("Please, enter your login and press Enter : ", "$/", 32);
  string password = getData("Please, enter your password and press Enter : ", "$/", 32);
  if (checkUser(fileMutex,"users.txt",login,password) == 0){
    ReleaseMutex(fileMutex);
    CloseHandle(fileMutex);
    if (championship == 2){
      passToCards();
      return 0;
    }
    passToGame(championship);
  } else {
```

```
    ReleaseMutex(fileMutex);
    CloseHandle(fileMutex);
    cout << "Invalid login or password" << endl;
  }
  return 0;
}

int checkUser(HANDLE fileMutex, const char* fileName, string login, string password){
  string rLogin;
  string rPassword;
  DWORD dwWaitResult = WaitForSingleObject(fileMutex,INFINITE);
  ifstream fin(fileName);
  int stop = 2;
  int del = 1;
  int index = 0;
  char ch;
  while ((stop!=0) && fin.get(ch)) {
    if (ch == '$'){
      stop--;
      if (stop == 0){
        stop = 1;
        del = 1;
        if ((login.compare(rLogin) == 0) && (password.compare(rPassword) == 0)) return
0;
        rLogin.clear();
        rPassword.clear();
      }
      continue;
    }
    if (ch == '/'){
      del--;
      continue;
    }
    if ((stop == 1) && (del == 1)){
      rLogin.append(1, ch);
      continue;
    }
    if ((stop == 1) && (del == 0) && (ch != '\n')){
      rPassword.append(1, ch);
      continue;
    }
  }
  if ((login.compare(rLogin) == 0) && (password.compare(rPassword) == 0)) return 0;
  fin.close();
  return 1;
}
```

2.2 Семафор

Для использования семафора была представлена ситуация, при которой могло бы стать необходимым ограничение числа пользователей, которые могут играть одновременно (как это делается в некоторых онлайн-играх для предотвращения «перегрузки» сервера). В данном случае имеем дело с задачей, в рамках которой требуется позволить выполнение программы только для определенного количества потоков исполнения, остальные же должны «встать в очередь» и ждать, пока кто-нибудь из игроков не покинет сервер. Трудно представить в данном случае более элегантное решение, чем введение семафора.

```
HANDLE getGameSemaphore(){
  HANDLE hSemaphore = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "gameSemaphore");
  if (hSemaphore == NULL){
    hSemaphore = CreateSemaphore(NULL, MAX_SEM_COUNT, MAX_SEM_COUNT, "gameSemaphore");
  }
```

```
  return hSemaphore;
}

int passToGame(int championship){
    HANDLE hSemaphore = getGameSemaphore();
    HANDLE hEvent;
    cout << "Waiting for queue..." << endl;
    DWORD dwWaitResult = WaitForSingleObject(hSemaphore, INFINITE);
    cout << "Welcome to the game!" << endl;
    if (championship == 1){
      hEvent = getChampionshipStartEvent(0);
      cout << "Waiting for starting championship..." << endl;
      WaitForSingleObject(hEvent, INFINITE);
    }
    int computer = 0;
    int user = 0;
    int num = 0;
    int computerNum = 0;
    string command;
    string comuputerCommand;
    int count = 0;
    int countComp = 0;
    srand(time(NULL));
    while (true){
        if ((championship == 1) && (WaitForSingleObject(hEvent,0) != WAIT_OBJECT_0)){
          cout << "Championship ended." << endl;
          CloseHandle(hEvent);
          break;
        }
        cout << "Let's go : \n - stone \n - scissors \n - paper \n - exit" << endl;
        getline(cin, command);
        if (command.compare("exit") == 0){
          break;
        }
        num = convertStringToStep(command);
        cout << num;
        if (num == -1) continue;
        computerNum = rand() % 3;
        comuputerCommand = convertStepToString(computerNum);
        cout << "You've selected " << command << " and I've selected " << comuputerCommand
<< endl;
        if (((num == 0) && (computerNum == 1)) || ((num == 1) && (computerNum == 2)) ||
((num == 2) && (computerNum == 0))){
          count++;
          cout << "You win. Count "<<count<<" : "<<countComp<<endl;
          continue;
        } else if (((num == 0) && (computerNum == 2)) || ((num == 1) && (computerNum ==
0)) || ((num == 2) && (computerNum == 1))){
          countComp++;
          cout << "I win. Count "<<count<<" : "<<countComp<<endl;
          continue;
        } else {
          cout << "Dead hit. Count "<<count<<" : "<<countComp<<endl;
          continue;
        }
    }
    cout << "Thank you for game!" << endl;
    cout << "End count "<<count<<" : "<<countComp<<endl;
    if (count > countComp){
      cout << "I win" << endl;
    } else if (count < countComp){
      cout << "You win" << endl;
    } else {
```

```
      cout << "Dead hit" << endl;
    }
    ReleaseSemaphore(hSemaphore, 1, NULL);
    CloseHandle(hSemaphore);
    return 0;
}
```

2.3 Событие синхронизации

Практически в любой мыслимой игре существуют чемпионаты. В данном случае было принято решение также ввести возможность проведения чемпионатов, однако возникла проблема: требуется, чтобы все игроки начинали и заканчивали игру одновременно, чтобы потом можно было сравнить количество выигрышей каждого из них и выбрать победителя с равными возможностями для всех. Очевидно, что необходим какой-либо арбитр, по разрешению которого чемпионат начинается или заканчивается. Для подобного арбитра как нельзя лучше подходит объект события синхронизации, который может быть либо в сигнальном состоянии (чемпионат идет), либо в несигнальном (чемпионат еще на начат или уже завершен).

```
HANDLE getChampionshipStartEvent(int access){
  HANDLE hEvent;
  if (access == 1){
    hEvent = OpenEvent(EVENT_ALL_ACCESS, FALSE, "championshipStartEvent");
  } else {
    hEvent = OpenEvent(SYNCHRONIZE, FALSE, "championshipStartEvent");
  }
  if (hEvent == NULL){
    hEvent = CreateEvent(NULL, TRUE, FALSE, "championshipStartEvent");
  }
  return hEvent;
}

int startChamp(){
  string keyword;
  HANDLE hEvent = getChampionshipStartEvent(1);
  if (WaitForSingleObject(hEvent,0) == WAIT_OBJECT_0){
    cout << "Championship already started" << endl;
    return 0;
  }
  do{
    cout << "Please, type keyword for starting championship : " << endl;
    getline(cin, keyword);
  } while (keyword.compare("hexademical") != 0);
  if (WaitForSingleObject(hEvent,0) == WAIT_OBJECT_0){
    cout << "Championship already started" << endl;
    return 0;
  }
  SetEvent(hEvent);
  if (WaitForSingleObject(hEvent,0) == WAIT_OBJECT_0){
    cout << "Championship have started !" << endl;
    return 0;
  }
  CloseHandle(hEvent);
  return 0;
}

int stopChamp(){
  string keyword;
  HANDLE hEvent = getChampionshipStartEvent(1);
  if (WaitForSingleObject(hEvent,0) != WAIT_OBJECT_0){
    cout << "Championship already stopped" << endl;
    return 0;
```

```
  }
  do{
    cout << "Please, type keyword for stopping championship : " << endl;
    getline(cin, keyword);
  } while (keyword.compare("hexademical") != 0);
  if (WaitForSingleObject(hEvent,0) != WAIT_OBJECT_0){
    cout << "Championship already stopped" << endl;
    return 0;
  }
  ResetEvent(hEvent);
  if (WaitForSingleObject(hEvent,0) != WAIT_OBJECT_0){
    cout << "Championship have stopped !" << endl;
    return 0;
  }
  CloseHandle(hEvent);
  return 0;
}

int passToGame(int championship){
    HANDLE hSemaphore = getGameSemaphore();
    HANDLE hEvent;
    cout << "Waiting for queue..." << endl;
    DWORD dwWaitResult = WaitForSingleObject(hSemaphore, INFINITE);
    cout << "Welcome to the game!" << endl;
    if (championship == 1){
      hEvent = getChampionshipStartEvent(0);
      cout << "Waiting for starting championship..." << endl;
      WaitForSingleObject(hEvent, INFINITE);
    }
    int computer = 0;
    int user = 0;
    int num = 0;
    int computerNum = 0;
    string command;
    string comuputerCommand;
    int count = 0;
    int countComp = 0;
    srand(time(NULL));
    while (true){
        if ((championship == 1) && (WaitForSingleObject(hEvent,0) != WAIT_OBJECT_0)){
          cout << "Championship ended." << endl;
          CloseHandle(hEvent);
          break;
        }
        cout << "Let's go : \n - stone \n - scissors \n - paper \n - exit" << endl;
        getline(cin, command);
        if (command.compare("exit") == 0){
          break;
        }
        num = convertStringToStep(command);
        cout << num;
        if (num == -1) continue;
        computerNum = rand() % 3;
        comuputerCommand = convertStepToString(computerNum);
        cout << "You've selected " << command << " and I've selected " <<
comuputerCommand << endl;
        if (((num == 0) && (computerNum == 1)) || ((num == 1) && (computerNum == 2)) ||
((num == 2) && (computerNum == 0))){
          count++;
          cout << "You win. Count "<<count<<" : "<<countComp<<endl;
          continue;
        } else if (((num == 0) && (computerNum == 2)) || ((num == 1) && (computerNum ==
0)) || ((num == 2) && (computerNum == 1))){
```

```
          countComp++;
          cout << "I win. Count "<<count<<" : "<<countComp<<endl;
          continue;
        } else {
          cout << "Dead hit. Count "<<count<<" : "<<countComp<<endl;
          continue;
        }
    }
    cout << "Thank you for game!" << endl;
    cout << "End count "<<count<<" : "<<countComp<<endl;
    if (count > countComp){
      cout << "I win" << endl;
    } else if (count < countComp){
      cout << "You win" << endl;
    } else {
      cout << "Dead hit" << endl;
    }
    ReleaseSemaphore(hSemaphore, 1, NULL);
    CloseHandle(hSemaphore);
    return 0;
}
```

2.4 Критическая секция

Помимо всего прочего, для данного случая была разработана модификация традиционной игры: два игрока по очереди выбирают любую карту из колоды, после чего сравнивают, что изображено на их картах и на основании этого выбирается победитель. После этого карты откладываются в сторону, и игра продолжается до тех пор, пока в колоде еще есть карты. Кроме того, оставшаяся колода перемешивается с постоянным интервалом времени для обеспечения большего интереса. Итак, встает задача: требуется ограничить игроков от выбора карт во время их перемешивания, иначе говоря, перемешивание карт не должно происходить одновременно с выбором двух карт. Это можно наиболее естественно реализовать с использованием объекта критической секции.

```
unsigned __stdcall reShuffleDeck(void* pThArg) {
  LPDWORD params = (LPDWORD)pThArg;
  int* sizePtr = (int*)params[0];
  int* deck = (int*)params[1];
  while(true){
    if (*sizePtr <= 0){
      _endthreadex(0);
    }
    EnterCriticalSection(&criticalSection);
    shuffleDeck(deck,*sizePtr);
    LeaveCriticalSection(&criticalSection);
    Sleep(1000);
  }
  return 0;
}

int passToCards(){
  srand(time(NULL));
  unsigned int threadId;
  cout << "Welcome to the stone, scissors and paper cards !" << endl;
  int* deck = generateDeck(10);
  int deckSize = 10;
  DWORD params[2];
  params[0] = (DWORD)&deckSize;
  params[1] = (DWORD)deck;
```

```
  HANDLE deckShuffler = (HANDLE)_beginthreadex(NULL, 0, reShuffleDeck, params,
CREATE_SUSPENDED, &threadId);
  ResumeThread(deckShuffler);
  int n;
  int m;
  int num;
  int computerNum;
  int count = 0;
  int countComp = 0;
  int tmp;
  while (deckSize > 0){
    cout << "Select card : " << endl;
    do{
      scanf ("%d",&n);
    }while ((n < 0) || (n >= deckSize));
    do{
      m = rand() % deckSize;
    } while (m==n);
    cout << "I select " << m << endl;
    EnterCriticalSection(&criticalSection);
    num = deck[n];
    computerNum = deck[m];

    deck[n] = deck[deckSize-1];
    deck[deckSize-1] = num;
    deckSize--;

    deck[m] = deck[deckSize-1];
    deck[deckSize-1] = computerNum;
    deckSize--;

    LeaveCriticalSection(&criticalSection);
    cout << "So, I've got "<<convertStepToString(computerNum)<<" and you've got
"<<convertStepToString(num)<<endl;
    if (((num == 0) && (computerNum == 1)) || ((num == 1) && (computerNum == 2)) || ((num
== 2) && (computerNum == 0))){
      count++;
      cout << "You win. Count "<<count<<" : "<<countComp<<endl;
      continue;
    } else if (((num == 0) && (computerNum == 2)) || ((num == 1) && (computerNum == 0))
|| ((num == 2) && (computerNum == 1))){
      countComp++;
      cout << "I win. Count "<<count<<" : "<<countComp<<endl;
      continue;
    } else {
      cout << "Dead hit. Count "<<count<<" : "<<countComp<<endl;
      continue;
    }
  }
  cout << "Thank you for game!" << endl;
  cout << "End count "<<count<<" : "<<countComp<<endl;
  if (count > countComp){
    cout << "I win" << endl;
  } else if (count < countComp){
    cout << "You win" << endl;
  } else {
    cout << "Dead hit" << endl;
  }
  return 0;
}

int main(int argc, char* argv[]){
  InitializeCriticalSection(&criticalSection);
```

```
  bool continueType = true;
  bool continueSubType = true;
  cout << "Welcome to the game!" << endl;
  string command = "";
  while(true){
    cout << "Menu : \n - login \n - signin \n - startchamp \n - stopchamp \n - about \n
- exit" << endl;
    continueType = true;
    while (continueType){
      getline(cin, command);
      if (command.compare("signin") == 0){
        cout << "Play mode : \n - plain \n - champ \n - cards \n - exit" << endl;
        continueSubType = true;
        while (continueSubType){
          getline(cin, command);
          if (command.compare("plain") == 0){
            admitUser(0);
            continueSubType = false;
          } else if (command.compare("champ") == 0){
            admitUser(1);
            continueSubType = false;
          } else if (command.compare("cards") == 0){
            admitUser(2);
            continueSubType = false;
          } else if (command.compare("exit") == 0){
            continueSubType = false;
          }
        }
        continueType = false;
      } else if (command.compare("startchamp") == 0){
        startChamp();
        continueType = false;
      } else if (command.compare("stopchamp") == 0){
        stopChamp();
        continueType = false;
      } else if (command.compare("login") == 0){
        registerUser();
        continueType = false;
      } else if (command.compare("about") == 0){
        cout << "---------System--Software-------\nIt's a sample of using multithreading
in C++ via WinAPI." << endl;
        continueType = false;
      } else if (command.compare("exit") == 0){
        cout << "Good buy!";
        ExitProcess(0);
      }
    }
  }
}
```

3. Вывод

Таким образом, в данной лабораторной работе были применены на практике знания об объектах синхронизации, области применения каждого из них и предоставляемых преимуществах. Безусловно, приведенная версия программы не может найти практического применения в том состоянии, в котором находится на момент сдачи лабораторной работы, однако впоследствии может быть значительно расширена при необходимости, на данный же момент она является хорошим наглядным примером использования объектов синхронизации, демонстрируя те случаи, в которых без них просто не обойтись.