Жүйелік бағдарламалау
ИП-22-6к1, 6к2, 6к3, 22-6тк
I деңгей сұрақтарына жауаптар:

---

1. Жүйелік бағдарламалық қамтамасыз етудің негізгі компоненттері

Жүйелік бағдарламалық қамтамасыз ету (ЖБҚ) – компьютердің аппараттық және бағдарламалық бөліктерін басқаруға арналған негізгі бағдарламалар жиынтығы. Оның негізгі компоненттері:

1. **Операциялық жүйе (ОЖ):**
   - Компьютердің барлық ресурстарын басқаруға және қолданушыларға ыңғайлы интерфейс ұсынуға жауап береді.
   - Мысалы: Windows, Linux, macOS.
2. **Драйверлер:**
   - Аппараттық құрылғылар мен ОЖ арасындағы байланыс орнатуға мүмкіндік беретін бағдарламалар.
   - Мысалы: принтер, видеокарта, желі адаптерлерінің драйверлері.
3. **Утилиталар:**
   - Жүйелік қызметтерді атқаратын қосымша бағдарламалар (дискіні тазалау, деректерді резервтік көшіру).
   - Мысалы: антивирус, архиватор.
4. **Қолдау кітапханалары (Libraries):**
   - Қолданбаларға арналған дайын функциялар жиынтығы. Windows жүйесінде бұл DLL файлдары ретінде беріледі.

---

2. Операциялық жүйе мен жүйелік бағдарламалар арасындағы айырмашылық

Сипаттама	Операциялық жүйе (ОЖ)	Жүйелік бағдарламалар
Міндеті	Аппараттық ресурстарды басқару, интерфейс ұсыну.	Жүйенің жұмысын кеңейту және қолдау.
Қолданушылармен байланыс	Тікелей байланыс орнатады.	Нақты тапсырмаларды орындау үшін қолданылады.
Мысалдары	Windows, Linux, macOS	Архиваторлар, диск утилиталары, антивирустық бағдарламалар, драйверлер.

Толықтыру:

- **Операциялық жүйе** (ОЖ) аппараттық ресурстарды басқару және пайдаланушы мен аппараттық жүйе арасындағы интерфейсті қамтамасыз етеді. ОЖ пайдаланушының бағдарламаларына ресурстарды бөлу, процессор уақытын басқару, жадты бөлу және басқа жүйелік ресурстарды басқаруды іске асырады.
- **Жүйелік бағдарламалар** жүйенің жұмысын қолдауға, әкімшілік етуге және кеңейтуге арналған. Бұл бағдарламалар көбінесе жүйенің жұмысын оңтайландыру, қауіпсіздікті қамтамасыз ету, файлдық жүйелерді басқару және жүйе деңгейінде басқа қызметтерді атқарады.

3. Win32 API дегеніміз не? Оның құрамына не кіреді?

Win32 API (Application Programming Interface) – Windows жүйесінде қолданбалар мен ОЖ арасындағы өзара әрекеттестікті қамтамасыз ететін бағдарламалық интерфейс.
Құрамына кіреді:

1. **Графикалық интерфейс (GDI):**
   - Графиканы басқару, бейнелерді, сызықтарды салу функциялары.
2. **Жадты басқару:**

- Бағдарлама үшін жад бөліп беру, оны босату.

3. **Файлдық жүйе:**
   - Файлдар мен каталогтармен жұмыс істеу функциялары.
4. **Ағындарды басқару:**
   - Жаңа ағындар құру, синхронизация функциялары.

---

4. Ағын ұғымы және оның түрлері

Ағын (Thread) – бағдарламада орындалатын дербес есептеу процесі. Бұл процессорлық уақытты бөлісу арқылы бағдарламаның бірнеше бөлігінің бір уақытта орындалуына мүмкіндік береді.

Ағын түрлері:

1. **Бірағынды (Single-threaded):**
   - Бағдарлама бір уақытта тек бір тапсырманы орындайды.
   - Мысалы: қарапайым калькулятор бағдарламасы.
2. **Көпағынды (Multi-threaded):**
   - Бағдарлама бірнеше тапсырманы бір мезгілде орындау үшін ағындарды кезекпен іске қосады.
   - Мысалы: музыканы ойнатып, мәтін теру бағдарламасы.
3. **Параллель ағындар (Parallel threads):**
   - Бірнеше ағын бір уақытта орындалады (көппроцессорлық жүйелерде).
   - Мысалы: мәліметтерді үлкен көлемде өңдеу.

---

5. Ағын бір күйден екінші күйге қалай ауысады?

Ағын күйлері:

1. **Running (жұмыс күйі):**
   - Ағын процессорда орындалып жатыр. Бұл күйде ағын өз жұмысын орындап жатыр.
2. **Waiting (күту күйі):**
   - Ағын ресурсты күтіп тұр. Мысалы, деректерді алу немесе сыртқы құрылғымен байланыс күтілуде.
3. **Ready (дайын күйі):**
   - Ағын орындалуға дайын, бірақ процессор бос емес. Бұл жағдайда ағын процессордың бөлінуін күтіп тұр.

Ауыстыру механизмдері:

- Ағындардың күйлерін ОЖ диспетчері басқарады. Диспетчер ағындардың күйлерін өзгертіп, ресурстарға қолжетімділікті басқаруды қамтамасыз етеді.
- Операциялық жүйе ағындарды басымдылыққа қарай орындатады. Егер бірнеше ағын дайын күйде болса, жүйе олардың басымдылығын ескеріп орындалатын ағынды таңдайды.

---

6. Ағындарды басқару түрлері

1. **Қолмен басқару:**
   - Бағдарламашы ағындарды тікелей басқарады. Бұл тәсілде программист ағындарды құру, тоқтату және басқаларын өз бетінше орындайды. Мысалы, арнайы кітапханалар немесе API арқылы ағындардың жұмысын басқару.
2. **Операциялық жүйе арқылы басқару:**
   - Операциялық жүйе ағындардың орындалу тәртібін автоматты түрде басқарады. Бұл тәсілде ОЖ өздігінен ағындарды жоспарлап, процессор уақытын бөледі, ағындарды кезекпен орындап, ресурстарды тиімді бөлуге көмектеседі.

---

7. Windows ОЖ ағындарының типтері

1. **Пайдаланушы ағындары (User Threads):**
    - Бұл ағындар бағдарламаның өзі немесе кітапханалар арқылы құрылады және басқарылатын болады. Пайдаланушы ағындары операциялық жүйенің ядросынан тәуелсіз жұмыс істейді және көбінесе жеңіл және жылдам жұмыс істеу үшін қолданылады.
2. **Ядролық ағындар (Kernel Threads):**
    - Бұл ағындар операциондық жүйенің ядросы арқылы басқарылып, жүйе ресурстарын тікелей пайдаланады. Ядролық ағындар көбінесе аппараттық ресурстарға тікелей қатынасу қажет болған кезде пайдаланылады. Оларға қатысты операциялық жүйе толық бақылауды жүзеге асырады.

---

8. Ағын прототипі қандай болады?

Ағынды құру үшін функцияның прототипі:

```
DWORD WINAPI ThreadFunction(LPVOID lpParameter);
```

- **DWORD**: Функцияның қайтару типі, бұл ағынның аяқталу кодын білдіреді.
- **WINAPI**: Windows операциялық жүйесінде қолданылатын функцияның шақыру конвенциясы.
- **ThreadFunction**: Ағынның негізгі жұмыс функциясы.
- **LPVOID lpParameter**: Ағынға берілетін параметр. Бұл параметр арқылы ағынның жұмыс істеуін баптауға болады.

---

9. Ағын қандай функция арқылы аяқталады?

Ағынды аяқтау үшін:

- **Функция:** `ExitThread()`

  Прототипі:

  ```
  void ExitThread(DWORD dwExitCode);
  ```

    - **dwExitCode**: Ағынның аяқталу коды. Бұл параметр ағынның аяқталу жағдайын білдіреді, ол басқа ағындарға немесе жүйеге хабарлануы мүмкін.

---

10. Ағын жұмысын тоқтату функциясы

Ағынды тоқтату үшін:

- **Функция:** `TerminateThread()`

  Анықтама: Бұл функция ағынды мәжбүрлі түрде тоқтатады. Бірақ оны қолдану ұсынылмайды, себебі ол ресурстарды босатуды қамтамасыз етпейді, бұл жүйе ресурстарының дұрыс босатылмауына және ағын жұмысын аяқтағаннан кейін ресурстардың «жоғалып кетуіне» әкелуі мүмкін.

  Прототипі:

  ```
  BOOL TerminateThread(HANDLE hThread, DWORD dwExitCode);
  ```

    - **hThread**: Тоқтатылуға тиіс ағынның дескрипторы.
    - **dwExitCode**: Ағын тоқтатылғаннан кейін қайтарылатын мән.

---

11. Ағын жұмысын қайта бастау үшін қандай функция қолданылады және оның прототипі қандай болады?

Ағынның жұмысын қайта бастау үшін Windows жүйесінде **`ResumeThread()`** функциясы қолданылады. Бұл функция бұрын тоқтатылған ағынды қайта іске қосады.

Қолдану мақсаты:

- Ағын уақытша тоқтатылып, кейін қажет болған жағдайда қайта іске қосылады. Бұл әдіс ресурстарды тиімді басқаруға мүмкіндік береді.

Прототип:

```
DWORD ResumeThread(HANDLE hThread);
```

Аргументтер:

- `hThread` – қайта басталатын ағынның дескрипторы.

Қайтарылатын мән:

- Егер функция сәтті орындалса, ағынның тоқтату саны қайтарылады.
- Егер `-1` қайтарылса, бұл қате орын алғанын білдіреді.

Мысалы:

```
HANDLE hThread = CreateThread(NULL, 0, MyThreadFunction, NULL,
CREATE_SUSPENDED, NULL);
if (hThread != NULL) {
    ResumeThread(hThread); // Ағынды қайта бастау
}
```

---

12. Псевдодескриптор мен дескриптордың айырмашылығы қандай?

Дескриптор (Descriptor):

- Бұл жүйелік объектілерді (мысалы, файлдар, ағындар, процестер) анықтау үшін қолданылатын нақты мән.
- Дескрипторлар жүйелік ресурстарды басқаруда маңызды рөл атқарады.

Псевдодескриптор (Pseudo-descriptor):

- Бұл ағымдағы процесс немесе ағын үшін арнайы жүйе ұсынатын мән. Ол тек белгілі контексте жарамды.
- Псевдодескрипторлар көбіне API деңгейінде жеңілдетілген басқаруды қамтамасыз етеді.

Мысал:

- `GetCurrentThread()` – ағымдағы ағынға псевдодескрипторды қайтарады.

Айырмашылықтар:

1. **Жарамдылық ауқымы:**
   - Дескриптор: Барлық процестер үшін жарамды.
   - Псевдодескриптор: Ағымдағы процесс шегінде ғана жарамды.
2. **Қолдану:**
   - Дескриптор: Жүйелік деңгейде ресурстарды басқарады.
   - Псевдодескриптор: Жұмысты жеңілдету үшін уақытша мән ретінде пайдаланылады.

---

13. Windows жүйесі приоритетінің класстарын атаңыз

Windows жүйесі процестердің басымдылығын анықтау үшін келесі приоритет кластарын қолданады:

1. **REALTIME_PRIORITY_CLASS:**
   - Ең жоғары басымдылық класы. Процестер нақты уақыт режимінде жұмыс істейді.
   - Мысалы, аппараттық сигналдарды өңдеу.
2. **HIGH_PRIORITY_CLASS:**
   - Жоғары басымдылық класы. Басқа процестердің орындалуын кідіртуі мүмкін.
   - Мысалы, бейне өңдеу бағдарламалары.
3. **ABOVE_NORMAL_PRIORITY_CLASS:**
   - Қалыптыдан сәл жоғары басымдылық.
   - Мысалы, фондық процестерге қарағанда маңызды тапсырмалар.
4. **NORMAL_PRIORITY_CLASS:**
   - Әдепкі басымдылық класы. Барлық стандартты бағдарламалар осы режимде жұмыс істейді.
5. **BELOW_NORMAL_PRIORITY_CLASS:**
   - Қалыптыдан төмен басымдылық.
   - Мысалы, аз маңыздылығы бар фондық процестер.
6. **IDLE_PRIORITY_CLASS:**
   - Ең төмен басымдылық.
   - Мысалы, жүйе бос болған кезде орындалатын тапсырмалар.

14. Ағындарды синхрондау дегеніміз не?

Ағындарды синхрондау – бірнеше ағынның бір ресурсты бір уақытта қолдану кезінде қақтығыстардың алдын алу процесі.

Қажеттілігі:

- Егер бірнеше ағын бірдей жадыға немесе құрылғыға қол жеткізсе, деректердің бұзылуы немесе жүйелік қателіктер туындауы мүмкін.
- Синхрондау ағындар арасындағы үйлесімділікті қамтамасыз етеді.

Синхрондау әдістері:

1. **Мьютекстер (Mutex):**
   - Бір ағынға ресурсты пайдалануға рұқсат береді.
   - Құлыпты ашқанша басқа ағын күту режимінде болады.
2. **Семафорлар (Semaphore):**
   - Бір уақытта бірнеше ағынға ресурсты пайдалануға мүмкіндік береді.
3. **Оқиғалар (Events):**
   - Белгілі бір шарт орындалғанда ағындарды басқаруға көмектеседі.
4. **Критикалық секциялар (Critical Sections):**
   - Тек бір ағынға қолжетімділікті қамтамасыз етеді, бірақ мьютекстерге қарағанда жылдамырақ.

---

15. Реестр дегеніміз не? Реестр арқылы басқару функциясы қандай қызмет атқара алады?

Реестр (Registry):

- Windows жүйесінде барлық конфигурациялық деректерді сақтайтын иерархиялық дерекқор.
- Оған операциялық жүйенің, драйверлердің және бағдарламалардың параметрлері кіреді.

Қызметтері:

1. **Параметрлерді оқу:** Бағдарлама немесе жүйе параметрлерін алу.
2. **Өзгерту:** Бағдарламалар немесе құрылғылар параметрлерін өзгерту.
3. **Жою:** Қажетсіз параметрлерді өшіру.

Функциялар:

- `RegOpenKeyEx()` – Реестр бөлімін ашу.
- `RegSetValueEx()` – Реестрге мән жазу.
- `RegDeleteValue()` – Мәнді өшіру.

---

16. Динамикалық байланысатын кітапхана дегеніміз не? Және оларды қолданудың артықшылығы қандай?

Динамикалық байланысатын кітапхана (DLL):

- Бұл Windows жүйесінде бірнеше бағдарлама бөлісе алатын сыртқы файлдар (.dll).
- Оларда код, деректер немесе ресурстар сақталады.

Артықшылықтары:

1. **Жадты үнемдеу:** Бірнеше бағдарлама бірдей DLL файлдарын қолданады.
2. **Жаңарту оңайлығы:** Бағдарламаны өзгертпей, тек DLL-ді жаңартуға болады.
3. **Модульдік принцип:** Бағдарламаны бірнеше модульдерге бөлуге мүмкіндік береді.

---

17. Процесстің негізгі атрибуттарын айтып бер

1. **Идентификатор (PID):** Процесске тағайындалатын бірегей сан.
2. **Жад:** Процесске бөлінген виртуалды жад көлемі.
3. **Ағындар:** Әрбір процесс құрамында бір немесе бірнеше ағын болуы мүмкін.

4. **Приоритет:** Процестің орындалу басымдылығы.
5. **Ресурстар:** Процесске бөлінген файлдар, құрылғылар және жады.

---

18. Ағын құрамына қандай компоненттер кіреді?
Ағын (Thread) құрамында бірнеше маңызды компоненттер бар, олар ағынның жұмысын және орындалуын қамтамасыз етеді:

1. **Регистерлер:**
   - Ағынның орындалу жағдайын сақтау үшін қолданылады. Әрбір ағын өзінің жеке регистрлері мен бағдарламалық санағышты (program counter) сақтайды, бұл ағынның орындау кезіндегі барлық күйін қайта қалпына келтіруге мүмкіндік береді.
2. **Жергілікті стек:**
   - Бұл ағынға арнайы бөлінген жад бөлігі, ол функциялар шақырулары, жергілікті айнымалылар және басқа да ағынға байланысты мәліметтерді сақтайды. Әр ағынның жеке стегі болады, бұл оның мәліметтерін басқа ағындардан оқшаулау үшін қажет.
3. **Приоритет:**
   - Ағынның жұмыс жылдамдығын анықтайды. Ағындардың орындалу тәртібі мен приоритеттері операциондық жүйе жоспарлаушысы арқылы реттеледі. Жоғары приоритетті ағындар алдымен орындалады.
4. **Контекст:**
   - Ағынның ағымдағы күйін сақтау үшін қажет. Контекстке регистерлер, стек және басқа да ағынға қатысты мәліметтер кіреді. Бұл контекст ағын жоспарлау кезінде өзгеріп отырады, және ағын тоқтатылғанда немесе орындалғаннан кейін контекст сақтау мен қалпына келтіру қажет болады.

---

19. Синхрондау объектілері деген не? Қандай класстарға жіктеледі?
Синхрондау объектілері: Ағындарды басқару және синхрондау үшін қолданылатын жүйелік объектілер. Олар бірнеше ағындар немесе процестер арасында ресурстарды бөлісу кезінде деректердің қауіпсіздігін қамтамасыз етеді. Синхрондау объектілері ағындардың өзара әрекеттесуін реттеп, оларды дұрыс тәртіпте орындауға мүмкіндік береді.
Класстар:

1. **Ядролық объектілер:**
   - **Мьютекстер (Mutexes):** Бір ағын ресурсты алғаннан кейін оны басқа ағындарға қолдануға болмайды, мьютекс қолдану арқылы осы ресурстарды қорғауға болады.
   - **Семафорлар (Semaphores):** Ресурстардың бірнешеуі бар болса, семафорлар ағындарға ресурстарды дұрыс бөліп береді.
   - **Оқиғалар (Events):** Ағындар арасында сигналдар жіберу үшін қолданылады, бір ағын белгілі бір жағдайдың орындалуын күтеді, ал екіншісі оқиғаны сигналдайды.
2. **Пайдаланушы деңгейіндегі объектілер:**
   - **Критикалық секциялар (Critical Sections):** Бір уақытта тек бір ағынға ғана ресурсты пайдалануға рұқсат беретін механизм. Бұл көбіне көп ағынды қосымшаларда деректерге қол жетімділікті басқару үшін қолданылады.

Синхрондау объектілері ағындардың үйлесімді және қауіпсіз түрде жұмыс істеуін қамтамасыз ету үшін өте маңызды, себебі олар көп ағынды бағдарламаларда деректердің бүтіндігін сақтайды.

---

20. Оқиғаларды басқаруда қандай функция қолданылады?
Функция: SetEvent()

Бұл функция оқиғаны сигналданған күйге ауыстырады. Оқиға сигналданған күйге ауысқанда, басқа процестер немесе ағындар осы оқиғаны күтуді тоқтатып, жұмысын жалғастыра алады. Оқиға объектілері жүйеде ағындар арасындағы синхрондауды қамтамасыз ету үшін қолданылады.

Прототип:

```
BOOL SetEvent(HANDLE hEvent);
```

Параметрлер:

- **hEvent:** Оқиға дескрипторы, оны `CreateEvent()` немесе басқа оқиға объектісімен басқару функциялары арқылы алған боларсыз.

Қайтару мәні:

- Егер функция сәтті орындалса, қайтару мәні `TRUE` болады.
- Егер функция қатемен орындалса, қайтару мәні `FALSE` болады, және қателік кодын алу үшін `GetLastError()` функциясын қолдануға болады.

Қолданылуы:

- Бұл функция негізінен ағындар мен процестер арасындағы синхрондауды қамтамасыз ету үшін қолданылады. Мысалы, бір ағын жұмысын тоқтатып, екінші ағынның белгілі бір жағдайды аяқтағаннан кейін жалғастыруын күтеді. `SetEvent()` оқиға объектісін сигналдай отырып, екінші ағынға бұл оқиғаның орындалғанын білдіреді.

21. Қандай функция каталогты орнатады?

Каталогты орнату үшін **`SetCurrentDirectory()`** функциясы қолданылады.

Прототип:

```
BOOL SetCurrentDirectory(LPCTSTR lpPathName);
```

Аргументтер:

- `lpPathName` – орнатылатын каталогтың жолы.

Қызметі:

- Бағдарламаның ағымдағы жұмыс каталогын өзгертеді.

Мысалы:

```
SetCurrentDirectory("C:\\MyDirectory");
```

---

22. Файлдың көшірмесін қандай функция арқылы аламыз?

Файлдың көшірмесін жасау үшін **`CopyFile()`** немесе **`CopyFileEx()`** функциясы қолданылады.

Прототип:

```
BOOL CopyFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, BOOL
bFailIfExists);
```

Аргументтер:

- `lpExistingFileName` – көшірілетін файлдың жолы.
- `lpNewFileName` – көшірменің жаңа атауы.
- `bFailIfExists` – егер файл бар болса, қате туындау керек пе.

Мысалы:

```
CopyFile("C:\\source.txt", "C:\\destination.txt", FALSE);
```

---

23. ExitProcess функциясының қызметі қандай?

`ExitProcess()` функциясы процесс жұмысын тоқтатады және барлық ағындарды аяқтайды.

Прототип:

```
VOID ExitProcess(UINT uExitCode);
```

Аргументтер:

- `uExitCode` – процесс аяқталу коды.

Қызметі:

- Барлық дескрипторларды жояды.
- Операциялық жүйеге процесс жұмысының тоқтағаны туралы хабарлайды.

Мысалы:
```
ExitProcess(0); // Процесс сәтті аяқталды
```

---

24. Файлдық жүйелердің түрлері және олардың параметрлерін айт

Файлдық жүйелердің негізгі түрлері:

1. **FAT (File Allocation Table):**
   - Жылдамдық: Жылдам, бірақ үлкен файлдарды басқаруға жарамсыз.
   - Шектеулер: 4 ГБ-қа дейінгі файлдарды ғана қолдайды.
2. **FAT32:**
   - Жақсартылған FAT нұсқасы.
   - Шектеулер: Бір файлдың максималды өлшемі 4 ГБ.
3. **NTFS (New Technology File System):**
   - Жетілдірілген қауіпсіздік және сығу мүмкіндіктері.
   - Шектеулер: Күрделірек құрылым.
4. **exFAT:**
   - Жақсы портативтілік үшін жасалған.
   - Шектеулер: Кейбір ескі құрылғылар қолдамайды.
5. **ReFS (Resilient File System):**
   - Деректердің сенімділігі мен үлкен деректер көлеміне арналған.
   - Шектеулер: Кейбір ескі Windows нұсқаларында қол жетімді емес.

---

25. Windows ОЖ қандай файлдық жүйелерді қолданады?

Windows операциялық жүйесі келесі файлдық жүйелерді қолданады:

1. **FAT/FAT32**:
   - **FAT (File Allocation Table)** және **FAT32** — ескі құрылғылар мен шағын тасымалдаушылар үшін кеңінен қолданылатын файлдық жүйелер. FAT32 файлдық жүйесі әдетте сыртқы қатты дискілерде, флеш-жадтарда және басқа шағын құрылғыларда пайдаланылады. Ол жүйенің қарапайымдылығымен танымал, бірақ кейбір шектеулері (мысалы, файлдың максималды өлшемі 4 ГБ) бар.
2. **NTFS (New Technology File System)**:
   - NTFS — Windows операциялық жүйесінің негізгі файлдық жүйесі. Бұл файлдық жүйе заманауи құрылғыларда кеңінен қолданылады, себебі ол жоғары өнімділікті қамтамасыз етеді және үлкен файлдар мен бөлімдерді қолдайды. NTFS файлдық жүйесі файл қауіпсіздігін, шифрлауды, журнал жүргізуді және басқа да көптеген мүмкіндіктерді ұсынады.
3. **exFAT (Extended File Allocation Table)**:
   - exFAT — FAT жүйесінің жаңартылған нұсқасы, ол үлкен файлдарды және үлкен көлемдегі тасымалдаушыларды қолдайды. exFAT, әдетте, флеш-жадтар мен портативті сақтау құрылғыларында (мысалы, SD карталарда) қолданылады. Бұл жүйе FAT32-нің шектеулерін жояды, бірақ NTFS-ке қарағанда жеңілірек.
4. **ReFS (Resilient File System)**:
   - ReFS — серверлер мен деректерді үлкен көлемде өңдеу үшін арналған файлдық жүйе. Бұл жүйе деректердің сақталуын қамтамасыз ету үшін коррекцияларды, автоматты қателік түзетуді және жоғары сенімділікті ұсынады. ReFS негізінен Windows Server операциялық жүйесінде пайдаланылады.

---

26. Файлдық жүйе атқаратын негізгі функция қандай?

Файлдық жүйенің негізгі функциясы – деректерді сақтау, ұйымдастыру және оларға жылдам қолжетімділікті қамтамасыз ету. Бұл функциялар әртүрлі әдістер мен құрылымдар арқылы жүзеге асырылады.

Міндеттері:

- **Файлдарды басқару (жасау, оқу, жазу, жою):**
  - Файлдық жүйе деректерді сақтау үшін файлдарды жасайды, оларды оқиды және жаңартады. Сонымен қатар, жүйе қажет емес файлдарды жоюды қамтамасыз етеді.
- **Метадеректерді сақтау (файл атауы, өлшемі, құқықтары):**
  - Файлдық жүйе әр файл үшін метадеректерді сақтайды. Бұған файлдың атауы, оның өлшемі, түрі, және қолжетімділік құқықтары (оқу, жазу, орындалу құқықтары) кіреді.
- **Қауіпсіздік және рұқсаттарды басқару:**
  - Файлдық жүйе файлдар мен каталогтар үшін рұқсаттарды және қауіпсіздік шараларын басқаруға мүмкіндік береді. Бұл пайдаланушылардың файлдарға қолжетімділігін реттеу және деректердің қорғауын қамтамасыз етеді. NTFS сияқты файлдық жүйелерде файлға қол жеткізу үшін рұқсаттар белгіленеді, бұл пайдаланушылардың әрқайсысының шектеулі мүмкіндіктерін анықтайды.

---

27. Реестрге қандай стандартты бөлімдер қызмет етеді?

Windows реестрі келесі стандартты бөлімдерден тұрады:

1. **HKEY_CLASSES_ROOT (HKCR):**
   - Файл типтері мен кеңейтімдеріне байланысты ақпаратты сақтайды.
2. **HKEY_CURRENT_USER (HKCU):**
   - Ағымдағы қолданушының параметрлері мен конфигурациясын сақтайды.
3. **HKEY_LOCAL_MACHINE (HKLM):**
   - Жүйенің барлық қолданушыларына арналған параметрлер.
4. **HKEY_USERS (HKU):**
   - Барлық қолданушылардың профильдеріне арналған деректер.
5. **HKEY_CURRENT_CONFIG (HKCC):**
   - Ағымдағы аппараттық конфигурация.

---

28. Файлдарды бұғаттау және бұғаттан алып тастауын қандай функция атқарады?

Файлдарды бұғаттау және бұғаттан алу үшін **`LockFile()`** және **`UnlockFile()`** функциялары қолданылады.

Прототиптері:

```
BOOL LockFile(HANDLE hFile, DWORD dwFileOffsetLow, DWORD dwFileOffsetHigh,
DWORD nNumberOfBytesToLockLow, DWORD nNumberOfBytesToLockHigh);
BOOL UnlockFile(HANDLE hFile, DWORD dwFileOffsetLow, DWORD dwFileOffsetHigh,
DWORD nNumberOfBytesToUnlockLow, DWORD nNumberOfBytesToUnlockHigh);
```

Мысалы:

```
LockFile(hFile, 0, 0, 100, 0);  // Файлды бұғаттау
UnlockFile(hFile, 0, 0, 100, 0); // Бұғаттан шығару
```

---

29. DLL қолдану арқылы қандай мүмкіндіктерге жол ашамыз?

DLL (Dynamic Link Library) қолдану арқылы:

1. **Кодты қайта пайдалану:** Әртүрлі бағдарламалар бірдей кодты бөлісе алады.
2. **Жадты үнемдеу:** Бірнеше процесс бір DLL-ді қолданады.
3. **Модульдік құрылым:** Бағдарламаны бірнеше бөліктерге бөлуге болады.
4. **Кітапхананы жаңарту:** Бағдарламаны өзгертпей, тек DLL-ді жаңартуға мүмкіндік бар.

30. DLL идентификатор дегеніміз не?

DLL идентификатор (Module Handle):

- Бұл DLL-ді жүйеде анықтайтын бірегей идентификатор.
- Оны **`GetModuleHandle()`** немесе **`LoadLibrary()`** функциялары арқылы алуға болады.

Мысалы:

```
HMODULE hModule = GetModuleHandle("MyLibrary.dll");
```

31. Windows-тағы ағын дегеніміз не? Оған қандай ресурстар жатады?

Ағын – бұл процесс ішінде орындалатын кодтың ең кішкентай бірлігі. Windows-та әрбір процесс кем дегенде бір ағыннан тұрады.

Ағын ресурстары:

1. **Регисторлар жиынтығы** – процессордың жұмыс күйін сақтайды.
2. **Стек** – функция шақырулары мен жергілікті айнымалыларды сақтау үшін қолданылады.
3. **Ағын идентификаторы** – ағынды басқа ағындардан ажырату үшін.
4. **Жұмыс күйі** – ағынның ағымдағы күйін (мысалы, жұмыс істеу немесе күту) анықтайды.
5. **Жоспарлау приоритеті** – ағынды орындау ретін анықтайтын деңгей.

32. Процесс және ағын арасындағы айырмашылықты түсіндіріңіз.

Сипаттама	Процесс	Ағын
Анықтама	Бағдарламаның орындалатын данасы.	Процесс ішіндегі орындалатын код бөлігі.
Жеке ресурстар	Өзінің жеке жад, файл және құрылғы дескрипторлары бар.	Процесс ресурстарын бөліседі.
Орындалу уақыты	Ауыр (құру мен жою ұзақ уақыт алады).	Жеңіл (тез құрылады және жойылады).
Кедергілер	Процесстер арасындағы ақпарат алмасу күрделі.	Ағындар арасында оңай дерек алмасады.

33. Операциялық жүйеде процестерді жоспарлау (scheduling) қалай жүзеге асырылады?

Процестерді жоспарлау – бұл процессор уақытын тиімді бөліп, процестерді орындау ретін анықтайтын операциялық жүйенің маңызды функциясы.

Жоспарлау алгоритмдері:

1. **FCFS (First Come First Serve):** Алдымен келген процесс бірінші орындалады.
2. **SJF (Shortest Job First):** Ең қысқа орындау уақытын талап ететін процесс бірінші орындалады.
3. **Round Robin (RR):** Процестерге уақытша квота бөлінеді.
4. **Priority Scheduling:** Жоғары приоритетті процесс бірінші орындалады.
5. **Multilevel Queue Scheduling:** Процестер бірнеше кезекке бөлінеді.

34. Қауіпсіздік дискрипторы қандай элементтерден тұрады?

Қауіпсіздік дискрипторы (Security Descriptor) Windows объектілерінің қауіпсіздік параметрлерін сипаттайды.

Негізгі элементтері:

1. **SIDs (Security Identifiers):** Қолданушылар мен топтардың идентификаторлары.
2. **DACL (Discretionary Access Control List):** Қолданушыға немесе топқа рұқсат беру немесе шектеу.
3. **SACL (System Access Control List):** Жүйе оқиғаларын бақылау.

4. **Owner:** Объектінің иесі.
5. **Group:** Объектінің топ идентификаторы.

---

35. Үймелерді жою функциясы және үймелерді жоюдың артықшылықтары
Үймелерді жою функциясы:

- **HeapDestroy()** — бұл функция динамикалық жадтан үйме (heap) аймағын жою үшін қолданылады. Бұл функция үймені толықтай босатып, барлық оған қатысты ресурстарды жүйеден жояды.

Прототип:

```
BOOL HeapDestroy(HANDLE hHeap);
```

- **hHeap** — үйме дескрипторы. Бұл параметр жоюға арналған үйменің дескрипторын көрсетеді.

Артықшылықтары:

1. **Жадты босатады, жад ағып кетуді болдырмайды:**
   - Үймені жою жадты дұрыс босатуға мүмкіндік береді, бұл жадтың ағу мәселесін болдырмайды. Егер үйме қолданыста болмаса, оны жою жадтың қосымша бөлінген бөлігіне қол жеткізуді тоқтатады.
2. **Жүйе өнімділігін арттырады:**
   - Үйме қолданудан шығып, оны жойғаннан кейін ресурстар босап, жүйе өнімділігі жоғарылайды. Бұл әсіресе үлкен көлемдегі динамикалық жадты қолданатын бағдарламаларда тиімді.
3. **Үйме қолданыссыз болса, қажет емес ресурстарды жоюға мүмкіндік береді:**
   - Егер үйме қажет емес болса немесе қолданылмаса, оны жою қажетсіз ресурстарды алып тастауға мүмкіндік береді. Бұл жүйеде орынды үнемдеп, оның жұмысын оңтайландырады.

Үймелерді дұрыс жою бағдарламаның тиімділігін сақтап, жүйе ресурстарын дұрыс басқаруға көмектеседі.

---

36. Оқиғаны дұрыс пайдаланбау неге әкеліп соқтырады?
Оқиғаны дұрыс пайдаланбау жүйеде бірнеше маңызды мәселелерге әкелуі мүмкін, олар бағдарламаның жұмысына теріс әсер етеді:

1. **Deadlock (Тұйықталу):**
   - Егер бірнеше процесс немесе ағын ресурстарға синхронсыз қол жеткізсе және бір-бірінің ресурстарын күтсе, жүйе тоқтап қалуы мүмкін. Оқиғаны дұрыс басқармау тұйықталуға (deadlock) алып келуі мүмкін, себебі процестер немесе ағындар бір-біріне тәуелді болады және ешқайсысы ресурстарды босатпайды.
2. **Race Condition (Жарыс жағдайы):**
   - Егер бірнеше ағын немесе процесс бір ресурсты бір уақытта синхронсыз пайдаланса, онда жарыс жағдайы (race condition) орын алады. Бұл жағдайда әр ағын немесе процесс ресурсты өңдеуде күтпеген нәтижелерге әкелуі мүмкін, мысалы, деректердің бұзылуы немесе дұрыс емес есептеулер.
3. **Қателіктер:**
   - Оқиғаны дұрыс басқармау қателіктерге алып келуі мүмкін, мысалы, жүйенің басқа компоненттері дұрыс сигналдар алмай қалады немесе оқиға шарттарын дұрыс интерпретацияламауы мүмкін. Бұл жағдайлар бағдарламаның дұрыс жұмыс істемеуіне, қателіктерге және өнімділіктің төмендеуіне себеп болуы мүмкін.

Оқиғаны дұрыс пайдаланбау бағдарламаның қауіпсіздігі мен тиімділігін бұзады, сондықтан оны дұрыс басқару өте маңызды.

---

37. fdwCreate жалаушалары

`fdwCreate` жалаушалары жаңа ағынды құру кезінде қолданылатын параметрлерді анықтайды. Ағын құру кезінде осы жалаушаларды орнату арқылы ағынның бастапқы күйі және оның жұмыс істеу ерекшеліктері көрсетіледі. Төменде `fdwCreate` жалаушаларының бірнеше мысалдары мен олардың мәндері:

1. **CREATE_SUSPENDED**:
    - Ағын күту күйінде басталады. Бұл жалауша орнатылғанда, жаңа ағын басталған кезде ол дереу орындалмайды. Ағынды тек арнайы команданың көмегімен іске қосуға болады. Әдетте бұл ағынды қосу алдында оның параметрлерін орнату қажет болғанда қолданылады.
2. **STACK_SIZE_PARAM_IS_A_RESERVATION**:
    - Стек өлшемі тек резерв ретінде белгіленеді. Бұл жалауша орнатылғанда, ағын үшін бөлінген стек физикалық жадта резервтеледі, бірақ нақты жад бөлу тек ағын жұмыс істей бастағанда орындалады. Бұл ағын үшін қажетті жад көлемін бақылауға мүмкіндік береді.

Осы жалаушалар ағынның басталуын және жұмыс істеу процесін басқаруға мүмкіндік береді, әсіресе ресурстарды үнемдеу және ағындарды тиімді басқару мақсатында.

---

38. fdwAttrsAndFlags қызметі және оның жалаушалары

`fdwAttrsAndFlags` параметрі файл немесе құрылғы дескрипторын құру кезінде қолданылатын әртүрлі параметрлерді анықтайды. Ол `CreateFile` функциясында файлды ашу кезінде оның атрибуттары мен мүмкіндіктерін орнату үшін пайдаланылады. Төменде `fdwAttrsAndFlags` параметрінің жалаушалары:

1. **FILE_ATTRIBUTE_READONLY**:
    - Файл тек оқуға арналған. Бұл жалауша файлды тек оқу үшін ашады, яғни файлға жазу әрекеттері орындалмайды. Егер файл жазуға арналса, онда бұл жалаушаны қолданбау керек.
2. **FILE_ATTRIBUTE_HIDDEN**:
    - Файл жасырын болып табылады. Бұл жалауша орнатылғанда, файл пайдаланушының көру мүмкіндігінен жасырын болады, бірақ файл әлі де жүйеде бар және оған қол жеткізуге болады.
3. **FILE_FLAG_OVERLAPPED**:
    - Асинхронды операциялар үшін. Бұл жалауша асинхронды операцияларды орындауға мүмкіндік береді. Яғни, файлмен жұмыс істеу кезінде басқа операциялар блокталмайды, бұл жүйе өнімділігін жақсартуға көмектеседі.
4. **FILE_FLAG_DELETE_ON_CLOSE**:
    - Файл жабылғанда файл автоматты түрде жойылады. Бұл жалауша орнатылғанда, файл жабылған кезде ол жүйеден автоматты түрде өшіріледі. Бұл, мысалы, уақытша файлдарды қолдану кезінде ыңғайлы болады.

Осы жалаушалар файлдың атрибуттарын және оның жұмыс істеу режимін анықтайды. Бұл параметрлер файлмен жұмыс істейтін бағдарламалардың дұрыс жұмыс істеуін қамтамасыз етуге көмектеседі.

---

39. Тәуелсіз үйінділерді қолдану қандай тиімділікті береді?

Тәуелсіз үйінділер (independent heaps) әрбір процесс немесе ағын үшін жеке жад бөлігін бөлу әдісін қолдану арқылы келесі артықшылықтарды береді:

1. **Жылдамдық**:
    - Әрбір процесс немесе ағын өз үймесін қолданған кезде, жадқа қол жеткізу жылдамырақ болады. Себебі жадты бөлу және басқару операциялары тек бір процесс немесе ағынға қатысты болады, бұл көп ағынды жұмыс кезінде тиімділік арттырады.
2. **Қауіпсіздік**:

- Тәуелсіз үйінділер әрбір процесс үшін оқшауланған жадтық кеңістікті қамтамасыз етеді, бұл бір процестің жадтық мәселелерінің немесе қателерінің басқа процестерге әсер етуінің алдын алады. Әрбір процесс өз жадын бақылап, тек өз ресурстарына қол жеткізе алады, бұл қауіпсіздік пен деректердің бүтіндігін қамтамасыз етеді.

3. **Оқшаулау**:
   - Әрбір процесс өзінің жадын қолданатындықтан, бір процестің жадын басқа процестерден оқшаулау мүмкіндігі артады. Бұл жадты басқаруды нақты бақылауға және әртүрлі процестер арасында үйлесімділікті қамтамасыз етуге мүмкіндік береді.

Тәуелсіз үйінділерді қолдану көптеген процестер мен ағындарды басқару кезінде маңызды артықшылықтар береді, әсіресе көп ағынды бағдарламалар мен көп процестерді басқаруға арналған жүйелерде.

---

40. Файлдарда уақытпен жұмыс жасауға арналған қандай файлдар бар?

Windows-тағы уақытпен жұмыс жасауға арналған функциялар:

1. **`GetFileTime()`** – файлдың уақыт белгілерін (жасау, өзгерту, кіру) алу.
2. **`SetFileTime()`** – файлдың уақыт белгілерін өзгерту.
3. **`FindFirstFile()`** – файлдың уақытын және басқа атрибуттарды іздеу.

---

41. Windows-тағы процесстер мен ағындардың айырмашылықтары

Сипаттама	Процесс	Ағын
Орындалу деңгейі	Тәуелсіз орындалады.	Процесс ішінде орындалады.
Жадты пайдалану	Өзінің жеке жад аймағы бар.	Барлық ағындар бір процестің жадын бөліседі.
Идентификатор	Процесске бірегей PID беріледі.	Ағынға TID беріледі.
Құру уақыты	Процессті құру қиынырақ және ұзақ.	Ағынды құру жылдамырақ.
Мақсаты	Негізгі бағдарлама жұмысы.	Процесс ішінде тапсырмаларды орындау.

42. Win32 жүйесіндегі процесстердің құрамында қандай компоненттер бар?

Win32 жүйесіндегі процесстердің негізгі компоненттері:

1. **Код сегменті:**
   - Бағдарлама кодын қамтиды.
   - Жадтың орындалатын бөлігі болып табылады.
2. **Деректер сегменті:**
   - Бағдарлама мәліметтерін (глобалды және статикалық айнымалылар) сақтайды.
3. **Стек (Stack):**
   - Функция шақырулары, жергілікті айнымалылар және қайтару адресі сақталады.
4. **Heap (Үйме):**
   - Динамикалық жад бөлу үшін қолданылады.
5. **Процесс идентификаторы (PID):**
   - Процессті басқа процестерден ажырату үшін бірегей идентификатор.
6. **Ағындар (Threads):**
   - Әрбір процесс кем дегенде бір ағыннан тұрады.
7. **Файл және құрылғы дескрипторлары:**

- Процесс ашқан барлық файлдар мен құрылғылардың идентификаторлары.

8. **Ресурстар:**
   - Модульдер (DLL файлдары).
   - Терезе объектілері, оқиғалар, таймерлер және т.б.

---

43. CreateProcess функциясының параметрлері

`CreateProcess` функциясы жаңа процесс құру үшін қолданылады.

Прототип:

```
BOOL CreateProcess(
  LPCSTR lpApplicationName,
  LPSTR lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL bInheritHandles,
  DWORD dwCreationFlags,
  LPVOID lpEnvironment,
  LPCSTR lpCurrentDirectory,
  LPSTARTUPINFO lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

Параметрлер:

1. **lpApplicationName:** Орындайтын файлдың аты.
2. **lpCommandLine:** Командалық жол параметрлері.
3. **lpProcessAttributes:** Процесстің қауіпсіздік атрибуттары.
4. **lpThreadAttributes:** Ағынның қауіпсіздік атрибуттары.
5. **bInheritHandles:** Процесстің дескрипторларын мұрагерлікке беру мүмкіндігі.
6. **dwCreationFlags:** Процесстің құру әдісін анықтайтын жалаушалар.
7. **lpEnvironment:** Процесстің қоршаған ортасы.
8. **lpCurrentDirectory:** Процесс үшін жұмыс каталогы.
9. **lpStartupInfo:** Процесстің бастапқы конфигурациясы.
10. **lpProcessInformation:** Процесстің ақпараттық құрылымы.

---

44. Бұғатталған функцияларды қалай тағайындаймыз?

Бұғатталған функциялар ресурстарды немесе ағындарды басқару үшін қолданылады.

Мысалы:

- Mutex, Semaphore, және Event объектілері арқылы ресурстарды бұғаттау.

Мысал:

```
HANDLE hMutex = CreateMutex(NULL, FALSE, "MyMutex");
// Ресурсты пайдалану алдында
WaitForSingleObject(hMutex, INFINITE);
// Ресурсты қолдану
ReleaseMutex(hMutex);
```

---

45. SetEvent және PulseEvent функциясының автобосатылған және қолмен босатылған оқиғамен айырмашылығы қандай?

1. **SetEvent:**
   - Оқиғаны сигналды күйге ауыстырады.
   - **Автобосатылған оқиға:** Бір ғана ағын босатылады.
   - **Қолмен босатылған оқиға:** Барлық күтіп тұрған ағындар босатылады.
2. **PulseEvent:**
   - Қол жетімді ағындарға сигнал жібереді, содан кейін оқиғаны қайта күту күйіне ауыстырады.

Ерекшеліктер:

- **Автобосатылған оқиға:** Бір ғана ағынның орындалуына мүмкіндік береді.
- **Қолмен босатылған оқиға:** Барлық күтіп тұрған ағындарды бірден белсендіреді.

46. Оқиға деген не? Және қандай түрлерге бөлінеді?

Оқиға (Event):
Процесстер мен ағындардың синхронизациясы үшін қолданылатын объект.
Түрлері:

1. **Автобосатылған оқиға (Auto-reset):**
   - Бір ағынды белсенді етеді.
   - Оқиға автоматты түрде күту күйіне оралады.
2. **Қолмен босатылған оқиға (Manual-reset):**
   - Барлық күтіп тұрған ағындарды белсенді етеді.
   - Қолмен күту күйіне қайтарылады.

---

47. Файлға қалай жазамыз?

Файлға жазу үшін келесі қадамдар орындалады:

1. **Файлды ашу:**
   - `CreateFile` функциясын қолдану.
2. **Жазу:**
   - `WriteFile` функциясын қолдану.
3. **Файлды жабу:**
   - `CloseHandle` функциясын қолдану.

Мысал:

```
HANDLE hFile = CreateFile("example.txt", GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
DWORD dwBytesWritten;
WriteFile(hFile, "Hello, World!", 13, &dwBytesWritten, NULL);
CloseHandle(hFile);
```

---

48. Файлдарды жабатын кезде қандай функция қолданылады?

Файлды жабу үшін Windows API-да `CloseHandle` функциясы қолданылады. Бұл функция файлға, құрылғыға немесе басқа объектілерге арналған дескрипторды жабуға мүмкіндік береді.

Синтаксис:

```
BOOL CloseHandle(HANDLE hObject);
```

Параметрлер:

- **`hObject`**: Жабылатын объектінің дескрипторы. Бұл параметр файлды, құрылғыны, немесе басқа Windows объектілерін білдіруі мүмкін. Ол `CreateFile` сияқты функциялардан алынған дескриптор болуы керек.

Қайтарылатын мән:

- Егер функция сәтті аяқталса, **`TRUE`** қайтарылады.
- Егер функция қате тапса, **`FALSE`** қайтарылады. Қателіктің себебін анықтау үшін `GetLastError` функциясын қолдануға болады.

Ескерту:

1. `CloseHandle` тек Windows объектілеріне қатысты қолданылады. Егер сіз басқа операциялық жүйеде жұмыс істесеңіз, басқа файлды жабу әдістерін қолдану керек.
2. Бір дескрипторды бірнеше рет жабуға болмайды. Егер бір дескрипторды екі рет жапсаңыз, бұл бағдарламада қателіктерге немесе күтпеген мінез-құлыққа әкелуі мүмкін.
3. Егер файл жабылмаса, онда файлға арналған жүйелік ресурстар босатылмауы мүмкін, бұл ресурс ағып кетуге әкеледі.

---

49. fdwFlags қандай мүмкіндіктерді анықтайды?

`fdwFlags` параметрі `CreateFile` функциясына файлды ашу кезінде қосымша параметрлерді беру үшін пайдаланылады. Бұл жалаушалар файлды ашу кезінде оның қасиеттерін немесе қандай операциялармен жұмыс істейтінін анықтайды. Төменде `fdwFlags` жалаушаларының кейбір маңызды мәндері:

1. **FILE_ATTRIBUTE_READONLY**:
   - Файл тек оқуға арналған. Файлды ашу кезінде жазу операциялары мүмкін болмайды.
   - Мысалы, егер файлды тек оқу үшін ашқыңыз келсе, бұл жалаушаны пайдаланыңыз.
2. **FILE_ATTRIBUTE_HIDDEN**:
   - Жасырын файл. Бұл жалауша файлды жасырын етіп көрсетеді, яғни ол пайдаланушы үшін стандартты файл шолғыштарында немесе командалық жолда көрсетілмейді.
3. **FILE_FLAG_OVERLAPPED**:
   - Асинхронды операциялар үшін. Бұл жалауша асинхронды (қатар орындауға мүмкіндік беретін) операцияларды орындау кезінде пайдаланылады. Файлмен жұмыс істеу кезінде басқа операцияларды күтуге мүмкіндік береді.
4. **FILE_FLAG_NO_BUFFERING**:
   - Буферлеуді өшіру. Бұл жалауша жүйенің файлға қол жеткізуін буферлеуді өшіреді, яғни барлық операциялар тікелей физикалық дискіге жасалады. Бұл үлкен файлдармен жұмыс істегенде өнімділікті жақсарта алады, бірақ белгілі бір шектеулер мен талаптарды орындау қажет.

Бұл жалаушалар файлды ашу кезінде қосымша конфигурацияларды орнатуға мүмкіндік береді, файлдың қасиеттері мен оның жұмыс істеу режимдерін нақтылауға көмектеседі.

---

50. CreateFile функциясы қандай қызмет атқарады?

`CreateFile` функциясы — Windows жүйесінде файлдарды ашу немесе жасау үшін қолданылатын негізгі функция. Ол сондай-ақ құрылғылармен байланыс орнату және жүйелік ресурстарға қолжетімділікті орнату үшін пайдаланылады. Бұл функция файлды немесе құрылғыны ашып, оған қол жеткізуге мүмкіндік береді, сонымен қатар файлмен әртүрлі операцияларды орындауға арналған қолжетімділік деңгейін орнатады.

Прототипі:

```
HANDLE CreateFile(
  LPCSTR lpFileName,                // Файлдың немесе құрылғының аты
  DWORD dwDesiredAccess,            // Қолжетімділік деңгейі (оқу, жазу және т.б.)
  DWORD dwShareMode,                // Файлды бөлісу режимі
  LPSECURITY_ATTRIBUTES lpSecurityAttributes, // Қауіпсіздік сипаттамалары
  DWORD dwCreationDisposition,      // Файлды жасау немесе ашу шарттары
  DWORD dwFlagsAndAttributes,       // Файлдың сипаттамалары мен атрибуттары
  HANDLE hTemplateFile              // Шаблон файл (копиялау үшін)
);
```

Мысал:

```
HANDLE hFile = CreateFile("example.txt", GENERIC_READ | GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
```

Мысал түсіндірмесі:

- `"example.txt"` — файлдың аты.
- `GENERIC_READ | GENERIC_WRITE` — файлға оқу және жазу құқықтарын беру.
- `0` — файлды басқа процесстермен бөлісу режимі (бұл жағдайда, ешқандай бөлісу жоқ).
- `NULL` — қауіпсіздік сипаттамаларының жоқтығы.
- `CREATE_ALWAYS` — егер файл бар болса, оны қайта жазу немесе жаңадан жасау.
- `FILE_ATTRIBUTE_NORMAL` — файлдың қалыпты атрибуттары (қосымша ерекше атрибуттарсыз).

- `NULL` — шаблон файл жоқ.

Қызметі:

- **Файлды ашу**: Егер файл бұрыннан бар болса, оны ашады.
- **Файлды жасау**: Егер файл жоқ болса, оны жасайды.
- **Құрылғымен байланыс**: Егер сіз файл емес құрылғыға (мысалы, портқа) қол жеткізгіңіз келсе, `CreateFile` сол құрылғымен байланысты орната алады.
- **Қолданушының қолжетімділігін реттеу**: Файлды немесе құрылғыны қандай режимде ашуға болатынын анықтайды (оқу, жазу және т.б.).

Бұл функция Windows жүйесінде файлмен немесе құрылғымен жұмыс істейтін көптеген операциялардың негізгі көзі болып табылады.

51. Динамикалық байланысатын кітапхана (DLL) не үшін қолданылады?

Динамикалық байланысатын кітапхана (DLL) — бұл белгілі бір функцияларды қамтамасыз ету үшін арнайы жасалған файл. DLL бағдарламалармен бірлесіп жұмыс істейді және бірнеше бағдарлама DLL кітапханасын бір уақытта қолдана алады. Бұл технологияның негізінде кодтың бір рет жазылып, көп рет қолданылуы жатыр. DLL жүйесі бағдарламалардың жалпы қолданылатын кодты пайдалануын қамтамасыз етеді.

Қызметтері мен артықшылықтары:

1. **Қайта қолданылатын код:**
   - DLL-дер бірнеше бағдарламалар үшін ортақ болуы мүмкін, бұл қайтадан код жазуды болдырмайды.
   - Бұрын жазылған және тексерілген кодты бірнеше бағдарламада қолдануға болады.
2. **Бағдарламаның көлемін азайту:**
   - DLL бағдарламаның бөлек модулі ретінде жұмыс істейді, сондықтан негізгі бағдарламада көлемді кодты сақтаудың қажеті жоқ. Бұл орындау файлының өлшемін азайтады.
3. **Жадты үнемдеу:**
   - DLL файлдары жүйеге бір рет жүктеледі және әртүрлі процестер арасында бөлісіліп қолданылады, яғни жадыны тиімді пайдалану мүмкіндігі бар.
4. **Бағдарлама жаңартулары:**
   - DLL кітапханасын жаңартқанда, бағдарламада арнайы өзгерістер енгізбей-ақ жаңартуларды жүзеге асыруға болады. Бұл үлкен жобаларда тиімді болады.
5. **Платформааралық жұмыс:**
   - DLL-дер өздері әртүрлі операциялық жүйелерде жұмыс істеуі мүмкін, сондықтан бағдарламаларды қайта жазу қажет емес.
6. **Параллель жұмыс:**
   - Бірнеше бағдарламалар немесе ағындар DLL-ді бір уақытта қолдана алады, бұл жүйені тиімдірек етеді.

Мысал:

DLL пайдалану үшін бірінші кезекте оны жүктеу керек. `LoadLibrary` функциясы арқылы жүктеледі, ал `GetProcAddress` функциясы арқылы белгілі бір функцияны шақыруға болады:

```
HINSTANCE hDLL = LoadLibrary("example.dll");
FARPROC func = GetProcAddress(hDLL, "FunctionName");
```

---

52. Windows жадысының типтері: физикалық және виртуалды жады деген не?

Физикалық жады:

- **Физикалық жады** деп компьютердің нақты жады құрылғылары (RAM) түсініледі.
- Бұл жүйенің негізгі жадын білдіреді, онда тек қана ағымдағы процестердің жұмыс деректері сақталады.

- Физикалық жадының көлемі шектеулі, ол компьютердің аппараттық бөлігінен тәуелді.

Виртуалды жады:

- **Виртуалды жады** - бұл жүйеде физикалық жадының көлемі шектеулі болған кезде қолданылатын технология.
- Ол физикалық жадының мөлшерін ұлғайтуға мүмкіндік береді, себебі виртуалды жады қатты дисктегі файлдар арқылы кеңейтіледі.
- Виртуалды жады әрбір процесс үшін жеке жеке бөлек адрестік кеңістігін қамтамасыз етеді, сондықтан бір процесс басқа процестің деректеріне қол жеткізе алмайды.
- Виртуалды жады жүйе өзінің жадын басқаруға мүмкіндік береді, онда әр түрлі бағдарламалар мен операциялық жүйе бөлек адрес кеңістігін пайдаланады.

---

53. Windows жадысын қорғаудың қандай әдістері бар?

1. **Адрестік кеңістікті бөлу:**
   - Әрбір процесс өз виртуалды адрестік кеңістігін пайдалануы тиіс. Әрбір бағдарламаға өзіне тиесілі жадтың бөлігі ғана қолжетімді болады. Бұл әдіс басқа процестердің деректерін бұзудың алдын алады.
2. **Data Execution Prevention (DEP):**
   - DEP технологиясы бағдарламалық қауіпсіздікті арттыру үшін қолданылады. Ол деректер сегментінде кодтың орындалуына жол бермейді, нәтижесінде зиянды кодтың орындалуы шектеледі.
3. **Address Space Layout Randomization (ASLR):**
   - Бұл технология бағдарламалық қауіпсіздікті арттыру үшін адрестік кеңістікті кездейсоқ орналастыру әдісі болып табылады. Осылайша, зиянды кодтың белгілі бір функциялар мен объектілердің орнын болжауы қиындайды.
4. **Windows Memory Protection:**
   - Windows жүйесінде физикалық жадпен жұмыс істеу кезінде түрлі қорғау әдістері қолданылады. Жадтың әрбір беті үшін рұқсат деңгейі анықталады (оқуға, жазуға, орындауға).
5. **Access Control Lists (ACL):**
   - Бұл тізімдер арқылы жадқа қолжетімділікті басқаратын жүйе. Олар қолданушыларға рұқсаттарды нақты анықтауға мүмкіндік береді.

---

54. Мьютекс пен семафордың артықшылығы мен кемшілігін айт

Мьютекс	Семафор
Артықшылықтары:	**Артықшылықтары:**
- Бір ресурсты тек бір ағын пайдалана алады.	- Бірнеше ағынға бір уақытта рұқсат берілуі мүмкін.
- Ағындар арасында синхрондауды қамтамасыз етеді.	- Орындалу ретіне басқару жүргізуге болады.
- Өте қарапайым және тиімді.	- Бірнеше ресурсты басқара алады.
Кемшіліктері:	**Кемшіліктері:**
- Ағын аяқталмайынша басқа ағын рұқсат ала алмайды.	- Артық блоктауды тудыруы мүмкін.
- Мутекс босатылмаған жағдайда, ресурстың қолданылуы шектеледі.	- Жоғары ресурстарды пайдалануы мүмкін.

---

55. Мьютекс пен семафораның айырмашылығы, олар не үшін қолданылады?

1. **Мьютекс:**
    - Мьютекс — бұл бір ағынға ғана қор ресурсын қолдануға мүмкіндік беретін объект. Оның артықшылығы — ресурсты тек бір ағын қолдана алады.
    - Мьютекс пайдаланылады, егер ресурс тек бір процесс немесе ағын үшін ғана бөлінсе.
    - Ол пайдаланылған ресурсты тек иесі босата алады.
2. **Семафор:**
    - Семафор — бұл белгілі бір шектеулер бойынша бірнеше ағындарға ресурстарды бөлісу мүмкіндігін береді.
    - Ол көп ағынды жүйелерде тиімді, себебі ол бірнеше ресурсты басқаруға мүмкіндік береді.
    - Семапор санын шектеу арқылы оның қалай бөлінетінін және кімге рұқсат етілетінін бақылауға болады.

---

56. LockFile және LockFileEx функциялары нені көрсетеді?

`LockFile`:

- Бұл функция файлдың белгілі бір бөлігін блоктауға мүмкіндік береді, осылайша басқа процестер оны оқи немесе жаза алмайды. Бұл файлмен бір уақытта тек бір процесс жұмыс істеуге мүмкіндік береді.

`LockFileEx`:

- Бұл функцияның айырмашылығы — ол бұғаттау кезінде қосымша параметрлерді (мысалы, асинхронды бұғаттау) реттеуге мүмкіндік береді.

Қолдану:
Егер бірнеше процесс бір файлды пайдаланатын болса, онда деректердің бүтіндігін сақтау үшін осы функциялар қолданылады.

---

57. Файлдарды бұғаттау кезінде қандай факторларды ескеру қажет?

1. **Файлдың бұғаттау диапазоны:**
    - Файлды бұғаттау кезінде оның қандай бөлігін бұғаттайтыныңызды білу маңызды. Егер файлдың тек бір бөлігі бұғатталса, қалған бөліктерге басқа процестер қол жеткізе алады.
2. **Рұқсат деңгейі:**
    - Файлды оқуға немесе жазуға рұқсат беретін деңгейлерді нақты анықтау қажет. Бұғаттау кезінде басқа процестерге тек оқуға немесе жазуға рұқсат беруі мүмкін.
3. **Процессаралық синхрондау:**
    - Бірнеше процестер арасында деректерді дұрыс синхрондау қажет. Бұғаттау әдістерін дұрыс таңдамау процесстің дұрыс жұмыс істемеуіне алып келуі мүмкін.

---

58. Файлдарды бұғаттау не үшін қажет және ең кең тараған бұғат функциясы?

Файлдарды бұғаттау қажет:

- Бірнеше процестің бір уақытта бір файлға қол жеткізуін болдырмау үшін.
- Деректердің бүтіндігін қамтамасыз ету үшін.

Ең кең тараған функциялар:

- **`LockFile`** — файлдың бөлігін бұғаттау үшін.
- **`UnlockFile`** — бұғаттау әрекетін аяқтау үшін.

---

59. FindFirstFile функциясының атқаратын қызметі қандай?

`FindFirstFile` — бұл функция каталогтағы файлдардың немесе ішкі каталогтардың тізімін алу үшін қолданылады. Бұл функция бірінші табылған файлды немесе каталогты қайтарады.

Прототип:

```
HANDLE FindFirstFile(LPCTSTR lpFileName, LPWIN32_FIND_DATA lpFindFileData);
``
`
```

Қызметі:

- Бұл функцияны қолдану арқылы каталогтағы барлық файлдарды немесе папкаларды табуға болады.

60. dwMoveMethod қандай режимді анықтайды?

`dwMoveMethod` параметрі Windows API-де файлды немесе басқа объектіні көшіргенде немесе оның позициясын өзгерткенде қолданылатын режимді анықтайды. Бұл параметр көбінесе **`SetFilePointer`** және **`MoveFile`** функцияларында кездеседі.

Мәндері:

`dwMoveMethod` параметрі үшін мүмкін болатын мәндер мыналар:

1. **`FILE_BEGIN`**
   Бұл мән көрсетілген позицияны файлдың басынан бастап есептейді.
   - Позиция көрсетілген орыннан бастап жылжыту үшін қолданылады.
2. **`FILE_CURRENT`**
   Бұл мән ағымдағы файл көрсеткішінен (курсордан) бастап есептейді.
   - Позиция ағымдағы позициядан кейін өзгертіледі.
3. **`FILE_END`**
   Бұл мән файлдың соңынан бастап есептейді.
   - Позиция файлдың соңынан кейін өзгертіледі.

Мысал:

`SetFilePointer` функциясында `dwMoveMethod` параметрі қолданылып, файл көрсеткішінің орнын өзгерту үшін пайдаланылатын мысал:

```
#include <windows.h>
#include <stdio.h>

int main() {
    HANDLE hFile = CreateFile("example.txt", GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile == INVALID_HANDLE_VALUE) {
        printf("Файлды ашу қатесі: %lu\n", GetLastError());
        return 1;
    }

    DWORD dwMoveMethod = FILE_END;  // Файлдың соңынан бастау
    DWORD dwNewPosition = SetFilePointer(hFile, 0, NULL, dwMoveMethod);

    if (dwNewPosition == INVALID_SET_FILE_POINTER) {
        printf("Файл көрсеткішін орнату қатесі: %lu\n", GetLastError());
        CloseHandle(hFile);
        return 1;
    }

    printf("Файл көрсеткіші сәтті өзгертілді.\n");

    CloseHandle(hFile);
    return 0;
}
```

Қысқаша сипаттамасы:

- **`FILE_BEGIN`**: Файлдың басынан бастап есептейді.
- **`FILE_CURRENT`**: Қазіргі позициядан бастап есептейді.

- **`FILE_END`**: Файлдың соңынан бастап есептейді.

Бұл параметр файлмен немесе объектілермен жұмыс істегенде қажетті қозғалыс немесе орналасуды басқаруға мүмкіндік береді.

61. Уақытпен жұмыс жасауға арналған қандай функциялар бар?

Windows жүйесінде уақытпен жұмыс істеуге арналған бірнеше функциялар бар, олардың негізгі мақсаты уақытты алу, орнату немесе салыстыру болып табылады.

1. **`GetSystemTime`**:
   - Жүйенің ағымдағы уақытын алу үшін қолданылады. Бұл функция әлемдік уақытты (UTC) береді.
2. **`GetLocalTime`**:
   - Жергілікті уақытты алу үшін пайдаланылады. Бұл функция жүйенің уақыт аймағына негізделген уақытты береді.
3. **`SetSystemTime`**:
   - Жүйенің уақытын орнатуға мүмкіндік береді. Бұл функция жүйелік уақытты (UTC) орнатады.
4. **`SetLocalTime`**:
   - Жергілікті уақытты орнату үшін қолданылады. Бұл функция жергілікті уақытты қайта орнатады.
5. **`GetTickCount`**:
   - Жүйенің іске қосылу уақытын миллисекундтармен береді. Бұл уақыт бағдарламаның жұмыс істеп жатқан уақытын анықтауға көмектеседі.
6. **`QueryPerformanceCounter` және `QueryPerformanceFrequency`**:
   - Жоғары дәлдікті уақыт өлшеу үшін қолданылады. Олар процессордың нақты уақытын өлшеуге мүмкіндік береді.
7. **`Sleep`**:
   - Бағдарламаны белгілі бір уақытқа тоқтатады. Бұл функцияда уақыт миллисекундтармен беріледі.
8. **`TimeGetTime`**:
   - Жүйенің ағымдағы уақытын миллисекундтармен қайтарады.
9. **`GetSystemTimeAsFileTime`**:
   - Жүйенің уақытын файл уақытының пішімінде алады.

---

62. Windows-тағы күту функциясы деген не? Оның қандай түрлері бар? HeapDestroy функциясы не үшін қолданылады?

Күту функциясы дегеніміз - процестің немесе ағынның белгілі бір уақытқа тоқтап, белгілі бір жағдай орын алғанша немесе белгілі бір ресурстар дайын болғанша күтуі үшін қолданылатын функциялар.

Күту функциялары:

1. **`WaitForSingleObject`**:
   - Бұл функция бір объектінің жағдайын күту үшін қолданылады (мысалы, мьютекс, оқиға, семафор). Ол объекті бос болғанда немесе уақыт шегіне жеткенде орындалады.
2. **`WaitForMultipleObjects`**:
   - Бірнеше объектінің жағдайын бір уақытта күту үшін қолданылады. Бұл функция бірнеше объектілердің бірі бос болған кезде немесе барлық объектілер орындалған кезде жұмыс істейді.
3. **`Sleep`**:
   - Бағдарламаны уақытша тоқтату үшін қолданылады.
4. **`SignalObjectAndWait`**:
   - Бұл функция бір объектіні белгілеу (сигнал беру) және басқа объектіні күту үшін қолданылады.

5. **`WaitForInputIdle`**:
    - Бір процесті немесе ағынды күту үшін қолданылатын функция, егер ол іске қосылғаннан кейін алғаш рет бос болса.

HeapDestroy:

- **`HeapDestroy`** функциясы динамикалық жадты басқару үшін пайдаланылады. Ол әдетте өзіңіз жасаған жадтың жиынтығын жою үшін қолданылады. Бұл функция тек сіз жасаған жадты жояды, бірақ стандартты жүйелік жадты жоймайды.

---

63. Жадты басқарудың негізгі әдістерін атаңыз.

1. **Сегменттеу (Segmentation)**:
    - Жадыны сегменттерге бөлу. Әр сегмент бір немесе бірнеше мәліметтер блогынан тұрады. Бұл әдіс бағдарламаның әрбір бөлігін жеке басқаруға мүмкіндік береді.
2. **Бетті бөлу (Paging)**:
    - Жадыны тең өлшемді беттерге бөлу. Бұл әдіс жадтың үздіксіз болуының қажеті жоқтығын көрсетеді, өйткені процесс виртуалды жадыда үздіксіз жұмыс істейді, бірақ ол физикалық жадыда басқа беттермен араласуы мүмкін.
3. **Қосымша бөлінген жад (Dynamic Memory Allocation)**:
    - `malloc`, `free`, `new`, `delete` сияқты функциялар арқылы жад бөліп алу мен босатуды жүзеге асыру.
4. **Heap немесе стек арқылы басқару**:
    - **Heap**: Бұл үлкен көлемдегі динамикалық жадты бөлу үшін қолданылады. Жад бөлу кезінде жадының көлемі динамикалық түрде өзгеруі мүмкін.
    - **Stack**: Бұл өте жылдам және шектеулі көлемдегі жад бөлу үшін қолданылады.

---

64. Виртуалды жад дегеніміз не және оның пайдасы қандай?

Виртуалды жад — бұл операциялық жүйенің және аппараттық құралдардың жадты пайдаланудың абстракциясы. Виртуалды жад физикалық жадының шектеулерінен тыс жадты пайдалану мүмкіндігін береді.

Пайдасы:

1. **Бағдарламалардың бөлек жұмыс істеуі**:
    - Әрбір процесс үшін жеке виртуалды адрестік кеңістік ұсынылады. Бұл бір процестің басқа процестің деректеріне қол жеткізуіне жол бермейді.
2. **Жадтың тиімді бөлінуі**:
    - Виртуалды жад арқылы барлық бағдарламалар үшін ортақ жадыны тиімді бөлуге мүмкіндік бар.
3. **Қосымша жад кеңістігі**:
    - Физикалық жадының мөлшеріне қарамастан, виртуалды жад үлкен болуы мүмкін, бұл бағдарламаларды кеңейтуге және оларды тиімді іске асыруға мүмкіндік береді.
4. **Жадыны бөлу және қорғау**:
    - Әр процесс тек өз кеңістігімен жұмыс істейді, басқа процестердің жадын бұзуы қиын.

---

65. Бос жадты бөлу әдістері қандай (мысалы, бөлек блоктар, беттер және сегменттер)?

1. **Блоктармен бөлу (Block Allocation)**:
    - Жад белгілі бір мөлшерде блоктарға бөлінеді. Әр блок белгілі бір мақсатқа, мысалы, файлға немесе деректерге арналған.
2. **Беттермен бөлу (Paging)**:

- Жадтың белгілі бір мөлшері (әдетте 4 Кб) «бет» деп аталады. Әр бет физикалық жадымен сәйкестендіріледі. Бұл әдіс үздіксіз емес жадты тиімді пайдалануға мүмкіндік береді.

3. **Сегменттермен бөлу (Segmentation)**:
   - Жад сегменттерге бөлінеді, және әр сегмент белгілі бір бағдарламалық құрылымға сәйкес келеді, мысалы, код сегменті, деректер сегменті және стектің сегменті.

---

66. Жадты бөлуде кездесетін мәселелерді атаңыз және оларды шешу жолдары қандай?

1. **Жадтың фрагментациясы**:
   - **Мәселе:** Жадты бөлгенде, әртүрлі процессорлық әрекеттерден кейін жадының кішігірім бөліктері бос қалады, бұл жаңа блоктардың бөлінуін қиындатады.
   - **Шешім:** Жадты қайта ұйымдастыру (defragmentation) немесе қосымша жад көлемін пайдалану.
2. **Шектелген физикалық жады**:
   - **Мәселе:** Физикалық жадының көлемі шектеулі, бірақ виртуалды жадыны бөлу қажет.
   - **Шешім:** Виртуалды жадты тиімді пайдалану, беттеу және сегменттеу әдістерін қолдану.
3. **Үлкен объектілерді бөлу**:
   - **Мәселе:** Үлкен объектілер үшін жады бөлуде қиындықтар туындауы мүмкін.
   - **Шешім:** Үлкен объектілерді бірнеше кішкентай блоктарға бөлу.

---

67. Win32-де қателерді өңдеу қандай функция арқылы жүзеге асады?

Win32 жүйесінде қателерді өңдеу үшін **`GetLastError`** функциясы пайдаланылады. Бұл функция соңғы орындалған операция кезінде туындаған қателікті анықтауға мүмкіндік береді.

Қолдану мысалы:

```
if (CreateFile(...) == INVALID_HANDLE_VALUE) {
    DWORD dwError = GetLastError();
    // Қателік кодын өңдеу
}
```

---

68. Файлдардың орналасуы туралы ақпаратты сақтау принциптері

Файлдардың орналасуы туралы ақпаратты сақтау үшін файлдық жүйе келесі принциптерді пайдаланады:

1. **Индекс кестелері:**
   Файлдардың нақты орны индекс кестесінде сақталады, онда әрбір файлдың аты мен оның орналасқан орны көрсетіледі.
2. **Файлдық жүйе блоктары:**
   Файлдың орналасуы белгілі бір блоктарда сақталады. Блоктардың көлемі әдетте 4 Кб немесе 8 Кб болады.
3. **Жол кестесі:**
   Әрбір файлдың жолы белгілі бір блоктардан тұрады, олар бір-бірімен сілтемелер арқылы байланысқан болуы мүмкін.

---

69. Файлдарды көрсету объектілерінің I кезеңі

Файлдарды көрсету объектілерінің бірінші кезеңі файлды ашу және оның негізгі сипаттамаларын анықтау үшін жауапты. Бұл кезеңде келесі қадамдар орындалады:

- **Файлды ашу**: Операциялық жүйе немесе бағдарлама файлды ашып, оның ішіндегі деректерге қол жеткізу үшін қажетті құрылымдарды құрады. Бұл кезеңде файлдың жолын көрсету, оның типін анықтау және дұрыс режимде ашу (оқу, жазу, қосымша режимдер) жүзеге асырылады.
- **Файл сипаттамаларын алу**: Файлдың көлемі, құрамы, құру уақыты, соңғы өзгеріс уақыты сияқты сипаттамалар жиналады. Бұл деректер файлмен жұмыс істеу барысында маңызды болып табылады.
- **Құрылымдардың орнатылуы**: Операциялық жүйе файлды тиімді басқару үшін қажетті құрылымдарды орнатады. Мысалы, файл дескрипторлары, буферлер және т.б. Бұл құрылымдар бағдарламаның файлмен жұмыс істеуін қамтамасыз етеді.

Бұл кезең файлды жүйеге енгізіп, оның бастапқы параметрлерін орнатуға жауапты.

---

70. Файлдарды көрсету объектілерінің II кезеңі

Файлдарды көрсету объектілерінің екінші кезеңі файлмен нақты жұмыс істеу үшін қажетті операцияларды орындауды қамтиды. Бұл кезеңде келесі әрекеттер орындалады:

- **Файлмен операциялар орындау**: Файлды оқу, жазу, өзгерту және т.б. операциялар орындалады. Бұл кезеңде бағдарламалар мен жүйе файлдағы деректермен жұмыс істейді. Оқуға арналған операциялар файлдың ішінен деректерді алады, ал жазуға арналған операциялар деректерді файлға қосады немесе оны жаңартады.
- **Файлды басқару**: Егер файл құрылымы немесе мазмұны өзгертілсе, бұл өзгерістер файлға енгізіледі. Мысалы, жаңа деректерді жазу кезінде файлдың көлемі немесе мазмұны өзгереді.
- **Файлды жабу**: Жұмыс аяқталғаннан кейін файлды жабу қажет. Бұл файлдың қорытынды күйін сақтап, жүйеде ресурстарды босатады. Қазіргі уақытта жабу процесі файлға жазылған деректердің дұрыс сақталуын және қажетті жабу операцияларын жүзеге асыруды қамтамасыз етеді.

Бұл кезең файлмен барлық операциялар аяқталып, оның тиімді жұмыс істеуі үшін қажетті әрекеттердің бәрі жүзеге асырылады.

71. DLL модулі қалай жасалады?

Dynamic Link Library (DLL) — бұл Windows жүйесінде қолданылатын және көптеген бағдарламалармен ортақ қолдануға болатын файл. DLL модулі жалпы кодты, ресурстарды (суреттер, мәліметтер, т.б.), және функцияларды бір файлда сақтайды, бұл көптеген бағдарламалардың осы кодты ортақ пайдалануына мүмкіндік береді.

DLL жасау үшін келесі қадамдарды орындау қажет:

1. **Жоба жасау**:
   - Visual Studio немесе басқа ортада жаңа DLL жобасын жасаңыз.
   - Жоба түрі ретінде **DLL** таңдаңыз.
2. **Функциялар мен ресурстарды қосу**:
   - DLL файлында қолданылатын функцияларды және ресурстарды анықтаңыз.
   - Әрбір функцияны экспорттау үшін `__declspec(dllexport)` атрибутын пайдаланыңыз.

   Мысалы:
   ```
   __declspec(dllexport) void MyFunction() {
       // Функцияның денесі
   }
   ```
3. **Файлды компиляциялау**:
   - Жобаны компиляциялап DLL файлын жасаңыз. Бұл файлды кейін басқа бағдарламаларда қолдануға болады.
4. **DLL файлының қолданылуы**:
   - DLL файлын бағдарламалық кодқа қосып, оған қатынасу үшін **`LoadLibrary`** және **`GetProcAddress`** функцияларын қолдануға болады.

72. Тақырыптық файл DLL файлының ішінде қалай қолданылады?

DLL-дің тақырыптық файлдары (`.h`) әдетте DLL-ді қолданатын бағдарламаларға DLL ішіндегі функциялар мен айнымалыларды көрсету үшін қолданылады. Тақырыптық файлдар функцияларды, құрылымдарды, айнымалыларды және басқа элементтерді анықтайды, сондықтан басқа кодтар DLL-ге қосылуға және оның қызметтерін пайдалануға мүмкіндік береді.

Тақырыптық файлды қолдану үшін қадамдар:

1. **DLL-дің тақырыптық файлын қосу**:
   - Тақырыптық файлды қосу үшін `#include` директивасын қолдану қажет.

   Мысалы:
   ```
   #include "mydll.h"
   ```
2. **Функцияларды жариялау**:
   - DLL-де қолданылатын функцияларды тақырыптық файлда жариялаңыз. Мұнда **`__declspec(dllexport)`** атрибуты қолданылып, функцияларды DLL ішінде экспорттауға мүмкіндік беріледі.

   Мысалы:
   ```
   __declspec(dllexport) void MyFunction();
   ```
3. **DLL қолданатын бағдарлама**:
   - DLL бағдарламасы мен оның тақырыптық файлдарын пайдаланатын бағдарлама олардың жұмысын орындау үшін **`LoadLibrary`** және **`GetProcAddress`** функцияларын қолдана алады.

---

73. Win32-де асинхронды енгізу-шығаруды орындау әдістері туралы не білесіз?

Асинхронды енгізу-шығару (I/O) — бұл процесс енгізу-шығару операциясы аяқталғанша басқа тапсырмаларды орындауға мүмкіндік береді. Win32 жүйесінде бірнеше асинхронды I/O әдісі бар:

1. **Асинхронды файлдық операциялар**:
   - **`ReadFile`** және **`WriteFile`** функциялары асинхронды түрде жұмыс істей алады. Бұл операциялар фондық ағындармен орындалады және шақыру барысында блокталмайды.
   - **`OVERLAPPED`** құрылымы: Асинхронды операцияларды басқару үшін қолданылатын құрылым.
2. **Асинхронды құбырлар**:
   - **`CreateNamedPipe`** және **`CallNamedPipe`** функцияларымен асинхронды құбырларды жасауға болады.
3. **Асинхронды желілік операциялар**:
   - **`WSASend`**, **`WSARecv`** сияқты Winsock функциялары асинхронды желілік операциялар үшін қолданылады.
4. **Windows хабарлары (Windows Messages)**:
   - Бағдарламаға асинхронды түрде кіріс деректерін алу үшін **`PostMessage`** және **`SendMessage`** функциялары арқылы хабарлар жіберуге болады.

---

74. Gonins және Gonouts дескрипторлары туралы айтып бер

Gonins және **Gonouts** дескрипторлары Windows жүйесіндегі енгізу-шығару операцияларын орындауға байланысты арнайы терминдер болып табылады. Олар көбінесе желілік байланыстармен жұмыс істегенде немесе процесстер арасындағы деректер алмасуында қолданылады. Алайда, бұл терминдер жиі кездесетін нақты сипаттамаларға ие емес, және ол нақты бағдарламалық контексте ерекшеленуі мүмкін.

---

75. FreeConsole және AllocConsole функциялары туралы айтыңыз

AllocConsole және **FreeConsole** функциялары Windows консоль режимінде қолданылатын арнайы функциялар болып табылады.

1. **AllocConsole**:
   - Консоль терезесін құрады және оны ағымдағы бағдарламаға тағайындайды. Бұл функцияны GUI қосымшаларында консоль енгізу-шығару функцияларын пайдалануға мүмкіндік береді.

   Мысалы:
   ```
   if (AllocConsole()) {
       // Консольды табыңыз
   }
   ```
2. **FreeConsole**:
   - Консоль терезесін жояды және ағымдағы процесс үшін оны босатады. Бұл функция консоль қосымшасын қажет емес кезде жою үшін пайдаланылады.

   Мысалы:
   ```
   FreeConsole();  // Консольды жабады
   ```

---

76. Стандартты құрылғылар әдетте қандай құрылғыларға тағайындалады?

Windows жүйесінде стандартты құрылғылар әдетте:

1. **stdin (Standard Input)** — әдетте пернетақта немесе басқа енгізу құрылғысына тағайындалады.
2. **stdout (Standard Output)** — экранға немесе басқа шығару құрылғысына тағайындалады.
3. **stderr (Standard Error)** — қателерді көрсету үшін экранға немесе файлға тағайындалады.

Бұл құрылғылар көбінесе қолданушыдан деректер алу және нәтижелерді көрсету үшін пайдаланылады.

---

77. Консольді енгізу-шығарудың арнайы функциялары туралы айтыңыз

Windows консольді енгізу-шығару үшін бірнеше арнайы функцияларды ұсынады:

1. **GetStdHandle**:
   - Стандартты енгізу-шығару құрылғыларының дескрипторларын алады.
2. **ReadConsole**:
   - Консольден деректерді оқуға арналған функция. Бұл функция асинхронды жұмыс істей алады.
3. **WriteConsole**:
   - Консольға деректерді жазуға арналған функция. Бұл функцияны нәтижелерді экранға шығару үшін қолдануға болады.
4. **SetConsoleTextAttribute**:
   - Консоль терезесінің мәтінінің түстерін өзгерту үшін қолданылады.

---

78. Асинхронды енгізу-шығарудың іске асыру тәсілдері қандай?

Асинхронды енгізу-шығаруды іске асыру үшін Windows бірнеше тәсілді ұсынады:

1. **Асинхронды операциялар (OVERLAPPED)**:
   - **ReadFile** және **WriteFile** сияқты функциялар асинхронды түрде орындалады, ол үшін **OVERLAPPED** құрылымы пайдаланылады.
2. **Құбырлар және желілік байланыс**:
   - **CreateNamedPipe** және **WSASend** сияқты асинхронды операцияларды қолдану.
3. **Windows хабарлары**:
   - Процесс немесе ағын хабарларды алу арқылы асинхронды түрде жұмыс істей алады.

79. Операциялық жүйелердің архитектуралық ерекшеліктері

Операциялық жүйелердің архитектуралық ерекшеліктері әртүрлі болуы мүмкін, бірақ бірнеше негізгі ерекшеліктері бар:

1. **Монолитті ядро** — ядроның барлық қызметтері бір бағдарламаның ішінде орналасады.
2. **Микроядро** — ядро өте шағын және жүйелік қызметтер бөлек процестер арқылы орындалады.
3. **Жұмыс процесстері мен ағындары** — процестер мен ағындардың басқарылуы.
4. **Қызметтердің басқарылуы** — жүйелік қызметтердің өзара әрекеттесуі.

---

80. API деген не? Және оның атқаратын қызметі қандай?

API (Application Programming Interface) — бұл бағдарламалық қосымшаларға өзара әрекеттесу үшін қажетті функциялар мен әдістер жиынтығы. API сыртқы кітапханалар мен жүйелік қызметтерге кіруді қамтамасыз етеді.

API атқаратын қызметтер:

- **Функцияларды ұсыну** — бағдарламаларға белгілі бір тапсырмаларды орындауға мүмкіндік береді.
- **Деректерді алмасу** — қосымшалар арасында деректер алмасуды жүзеге асырады.
- **Қызметтерді біріктіру** — түрлі жүйелер мен компоненттер арасындағы байланысты орнатады.

**81. О

перациялық жүйе құрылымын қандай элементтер құрайды?**

Операциялық жүйенің құрылымы келесі негізгі элементтерден тұрады:

1. **Ядро (Kernel)** — негізгі басқарушы жүйе.
2. **Драйверлер** — аппараттық құрылғыларды басқару үшін қажет.
3. **Қосымшалар** — пайдаланушыға қызмет көрсететін бағдарламалар.
4. **Қызметтер мен процестер** — жүйенің жұмысын басқару және іске қосу.

---

82. Ағынның салыстырмалы приоритетіне қандай класстар жатады?

Ағынның салыстырмалы приоритеті бойынша келесі класстар бар:

1. **Жоғары приоритетті ағындар** — жүйенің маңызды тапсырмаларын орындайды.
2. **Орташа приоритетті ағындар** — қалыпты операциялар үшін пайдаланылады.
3. **Төмен приоритетті ағындар** — фоновая жұмыс немесе пайдаланушының тапсырмалары үшін арналған.

83. ResumeThread функциясының қызметін түсіндір

`ResumeThread` функциясы Windows жүйесінде тоқтатылған (немесе үзіліс жасалған) ағынды қайта жандандырады. Бұл функция бұрын **`SuspendThread`** арқылы тоқтатылған ағынды қайта іске қосуға мүмкіндік береді.

Қызметі:

- **`ResumeThread`** тоқтатылған ағынды орындауды жалғастыру үшін қолданылады. Ағынды қайта жандандыру оның жұмысын қалпына келтіреді.
- Функция орындалған кезде жүйе бұрын тоқтатылған ағынның келесі командасын орындауға бастайды.

Мысалы:

```
HANDLE hThread = CreateThread(...);  // Ағынды құру
SuspendThread(hThread);              // Ағынды тоқтату
ResumeThread(hThread);               // Ағынды қайта іске қосу
```

---

84. Ағынның аяқталу функциясы мен прототипі туралы не білесіз?

Ағынның аяқталу функциясы — ағын жұмысын аяқтаған кезде орындалатын арнайы функция. Бұл функция ағын аяқталған кезде автоматты түрде немесе ағынның соңғы әрекетінде шақырылуы мүмкін.

Прототипі:

- Ағынның аяқталу функциясының прототипі келесідей:

```
DWORD WINAPI ThreadFunction(LPVOID lpParam);
```

Мұнда:

- **`ThreadFunction`** — ағынның орындаушы функциясы, ол ағын басталған кезде шақырылады.
- **`LPVOID lpParam`** — ағынға берілуі мүмкін параметр, оны функцияның ішінде пайдалану үшін қолданады.
- Функция аяқталған кезде **`ExitThread`** немесе **`return`** қолданылып ағын аяқталады.

Ағынның аяқталу процесі:

- Ағын аяқталған соң, ол жүйеде жойылады, және оның ресурстары босатылады.
- **`ExitThread`** функциясы ағынды аяқтау үшін қолданылады.

---

85. Процесс дегеніміз не? Оның өмірлік циклі қалай өтеді?

Процесс — бұл ресурстарды (есептеу қуаты, жады, енгізу-шығару құрылғылары) қолданатын бағдарлама немесе бағдарлама экземпляры.

Өмірлік цикл:

1. **Бастау (New)**: Процесс бастапқыда жүйе арқылы құрылады.
2. **Дайындық (Ready)**: Процесс орындалуға дайын, бірақ процессор күтіп тұр.
3. **Жұмыс істеп тұрған (Running)**: Процесс процессорды қолданады және командаларды орындайды.
4. **Күту (Waiting)**: Процесс ресурстарды (мысалы, файлға кіру) күтіп тұр.
5. **Аяқталу (Terminated)**: Процесс жұмысын аяқтайды және жүйе оны жояды.

---

86. Процесстерді жоспарлау (scheduling) қалай жүзеге асырылады?

Процесстерді жоспарлау — жүйеде көптеген процестер орындалып жатқанда процессор уақытын тиімді бөлу үшін жүзеге асырылатын процесс. Жоспарлау әдістері әртүрлі болуы мүмкін, бірақ негізгі түрлері мыналар:

1. **First-Come, First-Served (FCFS)** — бірінші келген процесс бірінші орындалады.
2. **Shortest Job First (SJF)** — ең қысқа уақытты қажет ететін процесс бірінші орындалады.
3. **Round-Robin (RR)** — әр процесс шеңбер бойынша уақыт бөлігін алады.
4. **Priority Scheduling** — әр процесс үшін приоритеттер белгіленеді, жоғары приоритетті процестер бірінші орындалады.

Жоспарлау барысында жүйе процесстердің кезектілігін және орындалуын тиімді басқару үшін арнайы алгоритмдерді қолданады.

---

87. Бірнеше ағынды (multithreading) бағдарламалау кезінде қандай мәселелер туындауы мүмкін?

Бірнеше ағынды бағдарламалау кезінде түрлі мәселелер туындауы мүмкін:

1. **Өзара әрекеттесу мәселелері (Concurrency Issues)**:
    - Бірнеше ағын бірдей деректерді өзгертуге тырысқанда деректердің бүлінуі мүмкін (race condition).
2. **Синхронизация мәселелері**:
    - Бірнеше ағынның бір уақытта жұмыс істеуі кезінде ресурстарға қол жеткізу тәртібі дұрыс болмаса, deadlock немесе livelock жағдайлары туындауы мүмкін.
3. **Ағындардың тиімділігі**:

- Ағындардың тым көп болуы жүйенің ресурстарын артық жүктеп, орындау тиімділігін төмендетуі мүмкін.

4. **Ағындар арасында байланыс**:
   - Ағындар арасында деректер алмасуды тиімді басқару үшін күрделі синхронизация қажет.

---

88. Процесстердің синхронизациясы дегеніміз не? Бұл қалай жүзеге асырылады?

Процесстердің синхронизациясы — бұл бірнеше процесстің өзара әрекеттесуін үйлестіру және деректерді қорғау мақсатында олардың әрекеттерін реттеу. Синхронизация қажет болғанда, процесстер бір уақытта жалпы ресурстарға қол жеткізуге тырысады.

Синхронизация әдістері:

1. **Мьютекстер (Mutex)**:
   - Мьютекс — бұл бір уақытта тек бір ағынға ресурс берілетін механизм.
   - `WaitForSingleObject` және `ReleaseMutex` сияқты функциялар мьютекс арқылы синхронизацияны жүзеге асырады.
2. **Семафорлар**:
   - Семафор — бұл ресурстарға қол жеткізу санын бақылау үшін қолданылады. Семафорлар ағындарға белгілі бір уақытта қанша ресурс бөлінетінін бақылауға мүмкіндік береді.
3. **Критикалық бөлімдер**:
   - Критикалық бөлімдер — бұл бірнеше ағын бір уақытта орындай алмайтын код фрагменттері.
4. **Атындар мен оқиғалар**:
   - Бұл синхронизация әдістері ағындарды белгілі бір оқиға немесе жағдай орындалғанға дейін күтуге мәжбүр етеді.

---

89. Microsoft ОЖ қандай топтарға бөлеміз? Және жүйенің қандай мүмкіндіктері бар?

Microsoft операциялық жүйелері бірнеше топқа бөлінеді:

1. **Windows NT ядросы**:
   - Бұл ядро жүйе ресурстарын басқару және қауіпсіздікті қамтамасыз ету үшін өте тиімді болып табылады.
   - Бұл жүйе жоғары өнімділік және тұрақтылықты қамтамасыз етеді.
2. **Windows 9x жүйесі**:
   - Ескі жүйелерге жатады, бірақ олар қарапайым қолданушыларға ыңғайлы интерфейс ұсынады.
3. **Windows Embedded**:
   - Бұл арнайы жүйелер үшін пайдаланылатын және аз ресурстарды талап ететін версия.

Microsoft ОЖ мүмкіндіктері:

- **Графикалық интерфейс**: Қолданушыға ыңғайлы интерфейс.
- **Қауіпсіздік**: Ресурстарға қол жеткізуді басқару, шифрлау, антивирус.
- **Мультимедиа**: Дыбыс, бейне, графикамен жұмыс жасау.
- **Мультизадачность**: Бірнеше процесті немесе ағынды бір уақытта басқару.

---

90. Ағындарды басқару алгоритмі қандай параметрлерді оптимизациялау үшін әзірленеді?

Ағындарды басқару алгоритмі келесі параметрлерді оптимизациялау үшін әзірленеді:

1. **Процессор уақытының тиімділігі**:
   - Ағындарға процессор уақытын тиімді бөлу.

2. **Жауап беру уақыты**:
    - Бағдарламаның пайдаланушыға жауап беру уақытын қысқарту.
3. **Тұрақтылық (Fairness)**:
    - Әрбір ағынға әділ түрде процессор уақытын беру.
4. **Ағындардың орындалуының тиімділігі**:
    - Ағындардың мүмкіндігінше көп жұмыс істеуі үшін ресурстарды тиімді пайдалану.
5. **Қателер мен кідірістерді болдырмау**:
    - Deadlock және басқа да ағынды мәселелерді болдырмау.

91. Қосымша функцияны немесе басқа DLL функциясын шақыру үшін қандай әрекет жасалады?

Қосымша функцияны немесе басқа DLL функциясын шақыру үшін келесі әрекеттер орындалады:

1. **DLL кітапханасын жүктеу**:
    - **`LoadLibrary`** функциясы арқылы DLL кітапханасы жүктеледі.
    - Бұл функция файл жолын немесе DLL атын қабылдайды және кітапхананың жадта орналасуын қамтамасыз етеді.

```
2. HMODULE hDll = LoadLibrary("mydll.dll");
```

3. **Функцияның адресін алу**:
    - **`GetProcAddress`** функциясы арқылы DLL ішіндегі функцияның адресі алынады.

```
4. FARPROC pFunc = GetProcAddress(hDll, "FunctionName");
```

5. **Функцияны шақыру**:
    - Функция адресі алынып, оны көрсету арқылы қажетті функция орындалады.

```
6. pFunc();  // немесе қажетті аргументтермен шақыру
```

7. **DLL кітапханасын босату**:
    - **`FreeLibrary`** функциясы арқылы кітапхана жадтан босатылады.

```
8. FreeLibrary(hDll);
```

---

92. Реестрмен жұмыс жасауға арналған API функцияларын айтып бер

Windows жүйесінде реестрмен жұмыс істеу үшін төмендегі API функциялары қолданылады:

1. **`RegOpenKeyEx`**:
    - Реестр кілтін ашуға арналған функция.

```
2. RegOpenKeyEx(HKEY hKey, LPCTSTR lpSubKey, DWORD ulOptions, REGSAM
   samDesired, PHKEY phkResult);
```

3. **`RegQueryValueEx`**:
    - Реестр мәнін оқу үшін қолданылатын функция.

```
4. RegQueryValueEx(HKEY hKey, LPCTSTR lpValueName, LPDWORD lpReserved,
   LPDWORD lpType, LPBYTE lpData, LPDWORD lpcbData);
```

5. **`RegSetValueEx`**:
    - Реестр мәнін жазу үшін қолданылатын функция.

```
6. RegSetValueEx(HKEY hKey, LPCTSTR lpValueName, DWORD Reserved, DWORD
   dwType, const BYTE* lpData, DWORD cbData);
```

7. **`RegCloseKey`**:
    - Реестр кілтін жабу үшін пайдаланылады.

```
8. RegCloseKey(HKEY hKey);
```

9. **`RegDeleteValue`**:
    - Реестрдегі мәнді жою үшін.

```
10.  RegDeleteValue(HKEY hKey, LPCTSTR lpValueName);
```

---

93. Үйінділермен жұмыс жасауға арналған API функцияларын айтып бер

Үйінділермен жұмыс істеу үшін Windows API-да келесі функциялар бар:

1. **`HeapCreate`**:

- Үйіндіні құру.

```
2. HANDLE HeapCreate(DWORD flOptions, SIZE_T dwInitialSize, SIZE_T
   dwMaximumSize);
```

3. **HeapAlloc**:
   - Үйіндіден жады бөлу.

```
4. LPVOID HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE_T dwBytes);
```

5. **HeapFree**:
   - Үйіндіден жадыны босату.

```
6. BOOL HeapFree(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem);
```

7. **HeapDestroy**:
   - Үйінді объектісін жою.

```
8. BOOL HeapDestroy(HANDLE hHeap);
```

---

94. Процесстермен жұмыс жасауға арналған API функцияларын айтып бер

Процесстермен жұмыс жасау үшін төмендегі API функциялары қолданылуы мүмкін:

1. **CreateProcess**:
   - Жаңа процесс құру үшін.

```
2. BOOL CreateProcess(LPCTSTR lpApplicationName, LPTSTR lpCommandLine,
   LPSECURITY_ATTRIBUTES lpProcessAttributes,
3.                    LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL
   bInheritHandles, DWORD dwCreationFlags,
4.                    LPVOID lpEnvironment, LPCTSTR lpCurrentDirectory,
   LPSTARTUPINFO lpStartupInfo, LPPROCESS_INFORMATION
   lpProcessInformation);
```

5. **TerminateProcess**:
   - Процесті аяқтау үшін.

```
6. BOOL TerminateProcess(HANDLE hProcess, UINT uExitCode);
```

7. **OpenProcess**:
   - Бар процесс бойынша дескриптор алу.

```
8. HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD
   dwProcessId);
```

9. **GetProcessId**:
   - Процестің идентификаторын алу.

```
10.  DWORD GetProcessId(HANDLE Process);
```

---

95. EXE мен DLL модулінің байланысы. Артықшылығы мен кемшілігін айтыңыз

EXE мен DLL арасындағы байланыс:

- **EXE** — бұл орындалатын файл, ол бағдарлама болып табылады.
- **DLL** (Dynamic Link Library) — бұл динамикалық түрде қосылатын кітапхана, ол қосымшаларға функционалдық мүмкіндіктер ұсынады, бірақ өздігінен орындалмайды.

Артықшылықтары:

- **DLL** файлдары кодтың қайта қолданылуын қамтамасыз етеді.
- **DLL** мүмкіндіктерін пайдалану арқылы EXE файлдары көлемін азайтады.
- Кодты бір рет жазады, ал әртүрлі бағдарламалар оны пайдалана алады.

Кемшіліктері:

- DLL мен EXE арасындағы үйлесімділік мәселелері туындауы мүмкін.
- DLL кітапханаларының дұрыс жүктелмеуі бағдарламаның істен шығуына әкелуі мүмкін.

---

96. DLL– да экспортталатын және импортталатын идентификатор деген не?

- **Экспортталатын идентификаторлар** — бұл DLL ішінде орналасқан функциялар мен айнымалылар, олар басқа қосымшалар немесе DLL-дер арқылы пайдалануға болады.
- **Импортталатын идентификаторлар** — бұл DLL-ға басқа бағдарламалар немесе DLL арқылы шақырылатын функциялар немесе айнымалылар.

Бұл идентификаторлар DLL-дың ішіндегі символдарды анықтайды, олар **`__declspec(dllexport)`** арқылы экспортталады және **`__declspec(dllimport)`** арқылы импортталады.

---

97. DLL қалай іске қосып, идентификаторларды қалай бір-бірімен байланыстырамыз?

1. **DLL жүктеледі**:
    - DLL кітапханасын жүктеу үшін Windows жүйесінде `LoadLibrary` функциясы қолданылады. Бұл функция DLL файлының жолын немесе атын қабылдайды, және оны жадқа жүктейді.

```
2. HMODULE hDll = LoadLibrary("mydll.dll");
3. if (hDll == NULL) {
4.     // Жүктеу сәтсіз болды
5. }
```

6. **Функцияның адресін алу**:
    - DLL ішіндегі нақты функцияның адресін алу үшін `GetProcAddress` функциясы пайдаланылады. Ол DLL ішінде көрсетілген атаумен функцияны табады және оның адресін қайтарады.

```
7. FARPROC pFunc = GetProcAddress(hDll, "FunctionName");
8. if (pFunc == NULL) {
9.     // Функция табылмады
10.  }
```

11. **Функция орындалады**:
    - Функцияның адресі алынып, шақыру үшін оны қолдануға болады. Бұл кезде функцияның параметрлерін дұрыс көрсету маңызды.

```
12.  // Функцияға сәйкес параметрлермен шақыру
13.  pFunc(arg1, arg2);
```

14. **DLL босатылады**:
    - DLL кітапханасын жадтан босату үшін `FreeLibrary` функциясы қолданылады. Бұл функция DLL жадтан жойылғаннан кейін, ресурстарды тазартады.

```
15.  FreeLibrary(hDll);
```

DLL арқылы бағдарламаның көлемін азайтып, қайталанатын кодтарды ортақ пайдалануға мүмкіндік береді. Сонымен қатар, DLL-дер бағдарламаның бірнеше бөліктеріне тәуелсіз түрде жаңартылуына жол ашады, бірақ DLL версиясының үйлесімділігі проблемасы туындауы мүмкін.

---

98. Файлдық жүйе дегеніміз не? Оның негізгі компоненттері қандай?

Файлдық жүйе — бұл ақпаратты сақтау және ұйымдастыру жүйесі. Ол дискінің физикалық кеңістігін логикалық түрде бөледі, файлдардың орналасуын бақылайды және файлдарға қолжетімділікті қамтамасыз етеді. Файлдық жүйелер деректерді құрылымды түрде сақтауға, оларды ұйымдастыруға және басқаруға мүмкіндік береді.

Негізгі компоненттері:

1. **Файлдар** — Файлдар — деректерді сақтаудың негізгі бірлігі. Әрбір файлдың аты, типі, және мазмұны бар. Файлдар текст, суреттер, бейнемазмұн немесе бағдарламалық код сияқты әртүрлі ақпараттарды сақтай алады.
2. **Каталогтар (немесе папкалар)** — Каталогтар файлдарды ұйымдастыруға арналған құрылымдар. Олар басқа каталогтарға сілтеме жасау арқылы ірі көлемді

деректерді жақсы ұйымдастыруға көмектеседі. Файлдық жүйеде каталогтар иерархиялық құрылымда болады.

3. **Жолдар** — Файлдың физикалық орналасу орнын немесе оның жүйедегі мекен-жайын көрсету үшін жолдар қолданылады. Олар дискінің түбірінен бастап, файлдың орналасқан жеріне дейінгі жолды көрсетеді.
4. **Файл дескрипторлары** — Файл дескрипторы — бұл файлды ашқан кезде жүйе тағайындайтын сәйкестендіруші (идентификатор). Ол файлмен әрекет жасағанда қолданылады, мысалы, оқу немесе жазу операцияларында.
5. **Метадеректер** — Файлдардың қасиеттерін (өлшемі, типі, соңғы өзгерту уақыты және т.б.) сипаттайтын ақпарат.

---

99. Операциялық жүйеде файлдармен жұмыс істеу тәсілдері қандай?

Операциялық жүйелер файлдарды басқарудың әр түрлі әдістерін ұсынады. Бұл тәсілдер қолданушылар мен бағдарламаларға файлдармен әрекет жасауға мүмкіндік береді.

Файлдармен жұмыс істеу тәсілдері:

1. **Файлды ашу**:
   - Файлды ашу үшін `CreateFile` немесе `fopen` сияқты жүйелік функциялар пайдаланылады. Файлдың атын, оны ашу режимін (оқу, жазу, қосу және т.б.) көрсету қажет.

```
2. HANDLE hFile = CreateFile("example.txt", GENERIC_READ | GENERIC_WRITE,
   0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

3. **Файлға жазу**:
   - Жазу операциясы `WriteFile` немесе `fprintf` арқылы жүзеге асырылады. Файлға деректер жазылады, және бұл деректер физикалық құрылғыда сақталады.

```
4. DWORD bytesWritten;
5. WriteFile(hFile, "Hello, World!", 13, &bytesWritten, NULL);
```

6. **Файлдан оқу**:
   - Файлдан оқу үшін `ReadFile` немесе `fscanf` қолданылады. Файлдың мазмұны жадыға көшіріледі.

```
7. DWORD bytesRead;
8. char buffer[100];
9. ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL);
```

10. **Файлды жабу**:
    - Файлды жапқаннан кейін оны қайта ашуға болмайды. Бұл операция `CloseHandle` немесе `fclose` арқылы жүзеге асады.

```
11.  CloseHandle(hFile);
```

---

100. Жүйелік бағдарламалау аясында деректерді сақтау үшін қолданылатын файлдық жүйелерді атаңыз.

Жүйелік бағдарламалау аясында әр түрлі файлдық жүйелер қолданылуы мүмкін, олардың әрқайсысының өз ерекшеліктері мен пайдалану жағдайлары бар.

Қолданылатын файлдық жүйелер:

1. **FAT (File Allocation Table)**:
   - FAT — ескі файлдық жүйе, әдетте флеш дискілерде және басқа портативті құрылғыларда қолданылады. Ол деректерді ұйымдастыру үшін қарапайым әдіс қолданады, бірақ қазіргі таңда үлкен файлдарды қолдауда шектеулер бар.
2. **NTFS (New Technology File System)**:
   - NTFS — Windows операциялық жүйесінде кеңінен қолданылатын файлдық жүйе. Бұл жүйе жоғары қауіпсіздік, үлкен файлдарды басқару, файлдардың кішірейтілген деректері мен метадеректерді сақтауды қамтамасыз етеді.
3. **ext4 (Extended File System 4)**:

- ext4 — Linux жүйесінде ең көп қолданылатын файлдық жүйе. Ол үлкен файлдарды қолдаумен қатар, жоғары өнімділік пен деректердің тұтастығын қамтамасыз етеді.

4. **HFS+ (Hierarchical File System Plus)**:
   - HFS+ — macOS жүйесінде қолданылатын файлдық жүйе. Ол үлкен файлдарды қолдау, файлдарды тез іздеу және қатты дискілерде тиімді жұмыс істеу үшін арнайы оңтайландырылған.
5. **exFAT (Extended File Allocation Table)**:
   - exFAT — FAT жүйесінің жетілдірілген нұсқасы. Ол үлкен көлемді сақтау құрылғыларында, мысалы, флеш-карталар мен сыртқы дискілерде қолданылады.

Бұл файлдық жүйелердің әрқайсысы өз ортасында тиімді, өйткені олар әр түрлі қажеттіліктер мен пайдаланушылардың талаптарына сай жобаланған.

101. Құрылғы драйвері дегеніміз не және оның қызметі қандай?

Құрылғы драйвері (device driver) — бұл операциялық жүйенің құрылғылармен өзара әрекеттесуін қамтамасыз ететін бағдарламалық компонент. Драйвер құрылғының физикалық немесе виртуалды аппараттық құралымен байланыс орнатады, оны басқаруды, оның жұмысын бақылауды және операциялық жүйемен ақпарат алмасуды жүзеге асырады.

Құрылғы драйверінің қызметтері:

1. **Құрылғыны басқару**:
   - Драйвер құрылғымен байланысты аппараттық ресурстарды басқарады. Мысалы, принтер драйвері принтердің жұмысын басқарады: оны қосу, басып шығару тапсырмаларын басқару және т.б.
2. **Құрылғының жағдайын бақылау**:
   - Құрылғының күйін тексеру және оның жұмыс жағдайын бақылау. Егер құрылғы жұмыс істемесе немесе ақау болса, драйвер қатені анықтап, жүйеге хабарлайды.
3. **Құрылғымен ақпарат алмасу**:
   - Драйвер құрылғы мен операциялық жүйе арасында деректер алмасуды қамтамасыз етеді. Бұл процесс құрылғының кіріс/шығыс операцияларын орындау кезінде орын алады.
4. **Операциялық жүйемен өзара әрекеттесу**:
   - Драйвер операциялық жүйеге құрылғының сипаттамаларын және оның мүмкіндіктерін хабарлайды. Сонымен қатар, жүйелік сұраныстарға сәйкес құрылғымен жұмыс істейді.
5. **Құрылғының физикалық деңгейіне тәуелсіз жұмыс**:
   - Драйвер операциялық жүйеге құрылғының нақты аппараттық ерекшеліктерін жасырады, яғни, әрбір құрылғының аппараттық бөлшектерінің жұмысын жүйеден жасырын етеді.

---

102. Операциялық жүйеде құрылғыларды басқару қалай жүзеге асырылады?

Операциялық жүйеде құрылғыларды басқару **құрылғы драйверлері** арқылы жүзеге асырылады. Құрылғыларды басқару бірнеше деңгейден тұрады:

1. **Құрылғы драйвері**:
   - Әрбір құрылғы үшін арнайы драйвер жазылады. Бұл драйверлер құрылғының жұмысын қамтамасыз етеді. Операциялық жүйе құрылғымен жұмыс істеген кезде драйверлер осы құрылғының аппараттық мүмкіндіктерін пайдаланады.
2. **Жоғары деңгейдегі басқару**:

- Операциялық жүйе құрылғыларды басқаратын жоғары деңгейлі функциялар мен бағдарламалар жиынтығын ұсынады. Мұнда құрылғыларды тану, оларды баптау және ресурстарды бөлу сияқты операциялар орындалады.

3. **I/O жүйелері**:
   - Құрылғыға енгізу-шығару операцияларын басқару үшін арнайы I/O жүйелері қолданылады. I/O жүйелері құрылғымен деректерді беру және қабылдау операцияларын ұйымдастырады.
4. **Ресурстарды бөлу**:
   - Операциялық жүйе құрылғыларды қажетті ресурстармен қамтамасыз етеді (мысалы, жад немесе процессор уақыты), бұл құрылғылардың жұмысын тиімді басқаруға мүмкіндік береді.
5. **Құрылғылармен байланыс**:
   - Операциялық жүйе мен құрылғылар арасындағы байланыс **перефириялық интерфейстер** арқылы жүзеге асады. Бұл интерфейстер арқылы құрылғылардың физикалық жадқа немесе процессорға қосылуы мүмкін.
6. **Планшеттер мен виртуализация**:
   - Операциялық жүйелер құрылғыларды виртуализациялауға мүмкіндік береді, яғни әртүрлі виртуалды құрылғыларды нақты аппараттық құрылғыларға тәуелсіз басқаруға мүмкіндік береді.

---

103. Драйверлер мен құрылғылар арасындағы өзара әрекеттесуді түсіндіріңіз.

Драйверлер мен құрылғылар арасындағы өзара әрекеттесу мынадай кезеңдерден тұрады:

1. **Құрылғыны іске қосу**:
   - Жүйе жұмысын бастамас бұрын драйверлер құрылғыны іске қосу үшін тиісті командаларды жібереді. Бұл кезеңде құрылғыны инициализациялау (жүктеу), оның аппараттық ресурстарын бөлу және драйвердің жұмыс істей бастауы орын алады.
2. **Құрылғының күйін тексеру**:
   - Құрылғы жүйеге қосылғанда, драйвер құрылғының күйін тексеріп, оның жұмыс істеп тұрғанын немесе ақаулардың бар-жоғын анықтайды.
3. **Деректерді өңдеу**:
   - Драйвер құрылғыдан немесе құрылғыға деректерді жібереді. Мысалы, дискіге жазу немесе принтерге басып шығару кезінде драйвер құрылғыға сұраныс жібереді және оған деректерді қайтарады.
4. **Құрылғымен байланыс**:
   - Құрылғы драйвері құрылғымен байланысты қамтамасыз етеді, құрылғының пәрмендерін операциялық жүйеге береді, сол арқылы жүйе құрылғымен тікелей жұмыс істейді.
5. **Құрылғыдан жауап алу**:
   - Құрылғы драйверге жауап қайтарады. Бұл деректерді өңдеу, ақаулар туралы хабарлама немесе құрылғының жағдайы туралы ақпарат болуы мүмкін. Драйвер бұл ақпаратты жүйеге жеткізеді.
6. **Құрылғының жұмысын тоқтату**:
   - Құрылғы тоқтаған кезде драйвер құрылғының жұмысын аяқтайды, барлық ресурстарды босатады және қажетті жабдықты өшіреді.

---

104. Жүйелік бағдарламалау үшін қандай тілдер қолданылады? Олардың ерекшеліктері қандай?

Жүйелік бағдарламалау аппараттық және операциялық жүйелермен жұмыс істейтін бағдарламаларды жазу процесін білдіреді. Бұл бағдарламалар көбінесе төмен деңгейдегі

бағдарламалау тілдерінде жазылады, себебі олар аппараттық деңгеймен тығыз байланысты.

Қолданылатын тілдер:

1. **C тілі**:
   - C тілі жүйелік бағдарламалау үшін ең танымал тілдердің бірі. Оның тиімділігі, портативтілігі және кең таралуы оны әртүрлі операциялық жүйелер мен құрылғыларды басқаруда өте танымал етеді.
   - **Ерекшеліктері**:
     - Төмен деңгейде жұмыс істеу мүмкіндігі.
     - Жүйелік ресурстарға тікелей қолжетімділік (жадқа және құрылғыларға).
     - Портативтілігі мен икемділігі.
     - Синтаксисі қарапайым және түсінікті.
2. **C++ тілі**:
   - C++ жүйелік бағдарламалауда кеңінен қолданылады, әсіресе объектілік-бағдарламалау принциптерін пайдалану қажет болғанда. Бұл тіл C тіліне негізделген, бірақ қосымша объектілік ерекшеліктерді енгізеді.
   - **Ерекшеліктері**:
     - ООП принциптерін қолдану мүмкіндігі.
     - Төмен деңгейдегі операцияларды жүзеге асыруға мүмкіндік береді.
     - Құрылғыға байланысты бағдарламаларды әзірлеуде кеңінен пайдаланылады.
3. **Assembly (Асемблер)**:
   - Асемблер тілі — бұл төмен деңгейдегі бағдарламалау тілі, ол процессордың машиналық командаларына жақын жұмыс істейді. Бұл тіл аппараттық құрылғылармен тікелей әрекеттесу үшін қолданылады.
   - **Ерекшеліктері**:
     - Тікелей аппараттық ресурстармен жұмыс істеу.
     - Жоғары жылдамдық пен тиімділік.
     - Құрылғыларды басқаруға арналған қолжетімді құралдар.
4. **Rust тілі**:
   - Rust тілі жүйелік бағдарламалауда соңғы жылдары кеңінен танылды. Оның артықшылығы қауіпсіздікке бағытталған бағдарламалауды қамтамасыз ету болып табылады.
   - **Ерекшеліктері**:
     - Жадты қауіпсіз басқару.
     - Жоғары өнімділік.
     - Көп ағынды бағдарламалауда оңай жұмыс істеу.
5. **Go тілі**:
   - Go тілі — бұл жүйелік бағдарламалау үшін қолданылатын жоғары деңгейлі тіл. Ол тиімділік пен қарапайымдықты қамтамасыз етеді, әсіресе серверлік және желілік бағдарламаларды құруда.
   - **Ерекшеліктері**:
     - Қарапайым синтаксис және жылдам компиляция.
     - Параллельді есептеулер мен көп ағынды бағдарламалауды жеңілдетеді.

---

105. C тілінің жүйелік бағдарламалаудағы рөлі қандай?

C тілі жүйелік бағдарламалау үшін негізгі тіл болып табылады, себебі ол аппараттық ресурстармен тиімді жұмыс істей алады, төмен деңгейлі құрылғылармен оңай байланысады және жақсы портативтілікке ие.

C тілінің рөлі:

1. **Жүйелік бағдарламаларды әзірлеу

**:

- Операциялық жүйелер, драйверлер, компиляторлар, жүйелік кітапханалар сияқты жүйелік бағдарламалар көбінесе C тілінде жазылады.

2. **Жоғары тиімділік**:
    - C тілі тиімді код жазуға мүмкіндік береді. Ол аппараттық ресурстарды тиімді пайдаланып, жүйелік ресурстарды оңай басқарады.
3. **Төмен деңгейде жұмыс істеу**:
    - C тілінде жазылған бағдарлама жүйелік ресурстарға, жадқа және құрылғыларға тікелей қолжетімділікке ие. Бұл жүйенің жұмысын оңтайландыруға мүмкіндік береді.
4. **Портативтілік**:
    - C тілінде жазылған бағдарламалар әртүрлі аппараттық платформаларда жұмыс істей алады, өйткені C тілінің синтаксисі қарапайым және әртүрлі операциялық жүйелер мен құрылғылармен үйлесімді.

C тілінің арқасында жүйелік бағдарламалау әлемі тиімді және ықшам болып, аппараттық және бағдарламалық жүйелер арасындағы байланыс нығайды.

106. Ассемблер тілінің қолдану аймағын түсіндіріңіз

Ассемблер тілі — бұл аппараттық ресурстармен тікелей жұмыс істеу үшін қолданылатын төмен деңгейлі бағдарламалау тілі. Бұл тіл машина командаларының негізінде жұмыс істейді және операциялық жүйелердің, драйверлердің, құрылғылардың жұмысын басқаруда маңызды рөл атқарады.

Ассемблер тілінің қолдану аймақтары:

1. **Операциялық жүйелерді дамыту**:
    - Ассемблер тілі операциялық жүйелердің ядросы мен құрылғыларды басқару бөлігін жазу кезінде пайдаланылады. Бұл тіл жүйелік қоңырауларды, interrupt өңдеушілерін және ресурстарды басқару операцияларын тиімді түрде жүзеге асырады.
2. **Құрылғы драйверлерін әзірлеу**:
    - Құрылғы драйверлерінде аппараттық ресурстармен тікелей әрекеттесу қажет болған кезде ассемблер тілі қолданылады. Ассемблер тілінде драйверлер құрылғының ерекшеліктерін ескеріп, аппараттық деңгейде жұмыс істейді.
3. **Жүйелік бағдарламалар**:
    - Жүйелік бағдарламалар мен утилиттер (мысалы, жүктеу жүктегіштер, компиляторлар) ассемблер тілінде жазылуы мүмкін, себебі ол тиімді және жылдам кодты қамтамасыз етеді.
4. **Анимация және графика**:
    - Ассемблер тілі графикамен жұмыс істегенде жоғары өнімділік қажет болған жағдайда қолданылады. Бұл әсіресе видеокарта мен басқа құрылғыларды басқару кезінде тиімді.
5. **Қауіпсіздік және қорғаныс бағдарламалары**:
    - Ассемблер тілінде жазылған бағдарламалар операциялық жүйенің қорғаныс механизмдеріне, вирусқа қарсы бағдарламаларға және қауіпсіздік утилиттеріне қолданылады.
6. **Микроконтроллерлер және embedded жүйелер**:
    - Микроконтроллерлер мен embedded жүйелерде ассемблер тілі кеңінен қолданылады, себебі ол жүйелік ресурстарды тиімді басқаруға мүмкіндік береді.

---

107. Жүйелік бағдарламалау саласында қолданылатын негізгі деректер құрылымдары қандай?

Жүйелік бағдарламалауда әртүрлі деректер құрылымдары қолданылады, себебі олар аппараттық ресурстарды тиімді пайдаланып, әртүрлі тапсырмаларды орындауға мүмкіндік береді. Бұл құрылымдар көбінесе төмен деңгейде жұмыс істейтін бағдарламаларда, мысалы, операциялық жүйелерде, драйверлерде және басқа жүйелік бағдарламаларда пайдаланылады.

Негізгі деректер құрылымдары:

1. **Массивтер**:
   - Массивтер бір типті деректердің жиынтығын сақтайды және оларды индекстер арқылы қолжетімді етеді. Бұл құрылым жиі оперативтік жадыны басқаруда және үлкен деректерді өңдеуде қолданылады.
2. **Жиындар (Sets)**:
   - Жиындар қайталанбайтын элементтерден тұрады. Олар жиі мәліметтер базасында немесе деректерді сұрыптау мен іздеуде қолданылады.
3. **Тізбектер (Lists)**:
   - Тізбектер элементтерді тізбектей сақтау үшін қолданылады. Олар динамикалық жадта орын бөлу мүмкіндігін қамтамасыз етеді және элементтерді оңай қосып немесе өшіруге мүмкіндік береді.
4. **Стектер (Stacks)**:
   - Стектер LIFO (Last In, First Out) принципі бойынша жұмыс істейді, яғни соңғы енгізілген элемент бірінші өңделеді. Олар функциялар шақырулары, жадты басқару және қалпына келтіру үшін қолданылады.
5. **Кезектер (Queues)**:
   - Кезектер FIFO (First In, First Out) принципі бойынша жұмыс істейді. Бұл құрылым көбінесе мәліметтерді өңдеу немесе ресурстарды басқару кезінде қолданылады.
6. **Ағаштар (Trees)**:
   - Ағаштар деректерді иерархиялық түрде сақтайды және сұрыптау, іздеу, жою операцияларын тиімді жүзеге асыру үшін қолданылады. Олар операциялық жүйелерде процестерді басқаруда және ресурстарды бөлу кезінде пайдаланылады.
7. **Графтар (Graphs)**:
   - Графтар деректерді нодалар мен байланыстырушы шеттер түрінде сақтайды. Графтар желілік құрылымдар мен деректер арасындағы қатынастарды бейнелеу үшін қолданылады.
8. **Хэш кестелер (Hash Tables)**:
   - Хэш кестелері элементтерді жылдам іздеу үшін қолданылады. Олар операциялық жүйелерде процесс идентификаторларын немесе файл жүйесін басқаруда жиі кездеседі.

---

108. Құрылғы драйверлерінде қолданылатын алгоритмдер қандай?

Құрылғы драйверлерінде қолданылатын алгоритмдер аппараттық құрылғылармен байланыс орнату, деректерді өңдеу және құрылғының жұмысын басқару үшін пайдаланылады. Әрбір құрылғының ерекшеліктеріне байланысты әртүрлі алгоритмдер қолданылуы мүмкін.

Қолданылатын негізгі алгоритмдер:

1. **Блокты және ағынды енгізу-шығару алгоритмдері**:
   - Бұл алгоритмдер құрылғымен деректер алмасуды басқару үшін қолданылады. Олар мәліметтерді блоктап немесе ағын түрінде жібереді, мысалы, қатты диск немесе желілік құрылғылармен жұмыс істеуде.
2. **Interrupt басқару алгоритмдері**:

    - Құрылғылардан келген үзілістерді басқару үшін арнайы алгоритмдер қолданылады. Interrupt сұранысы келген кезде драйвер жүйеге қайтарым жасайды және қажетті әрекеттерді орындайды.
3. **Деректерді буферлеу (Buffering)**:
    - Бұл алгоритм құрылғымен деректерді өңдеу кезінде олардың жоғалуын болдырмау үшін қолданылады. Буферлер деректерді аралық сақтауда қолданылып, олардың кезекпен өңделуіне мүмкіндік береді.
4. **Үзіліс өңдеушілер (Interrupt Handlers)**:
    - Бұл алгоритмдер аппараттық құрылғылардан келген үзілістерді тиімді өңдеу үшін пайдаланылады. Әрбір құрылғы үшін арнайы үзіліс өңдеушісі жасалады.
5. **Ресурстарды бөлу алгоритмдері**:
    - Драйверлер құрылғылардың ресурстарын бөлу үшін арнайы алгоритмдер қолданады. Мысалы, принтер немесе процессор уақытын бөлуде.
6. **Қатені түзету алгоритмдері**:
    - Құрылғы драйверлері қателерді түзету және қатені анықтау алгоритмдерін пайдаланады. Олар жүйе мен құрылғы арасындағы деректерді қателіксіз өңдеуге көмектеседі.

---

109. Жүйелік бағдарламалауда ағаштар, графтар және тізбектер қандай мақсатта пайдаланылады?

Жүйелік бағдарламалауда ағаштар, графтар және тізбектер деректерді тиімді басқару үшін маңызды рөл атқарады. Олар әртүрлі жүйелік және қосымша бағдарламаларда түрлі мәселелерді шешу үшін қолданылады.

Қолдану мақсаттары:

1. **Ағаштар**:
    - Ағаштар жүйелік бағдарламалауда деректерді сұрыптау, индекстеу және иерархиялық құрылымдармен жұмыс істеу үшін қолданылады. Операциялық жүйелердегі процесс басқару, файлдық жүйелер мен виртуалды жадыны басқару үшін пайдаланылуы мүмкін.
2. **Графтар**:
    - Графтар жүйелік бағдарламалауда желілерді модельдеу, деректер арасындағы қатынастарды көрсетуде және жоспарлау мәселелерін шешуде қолданылады. Графтарды операциялық жүйелердегі процестерді синхрондау және ресурстарды бөлу үшін пайдалануға болады.
3. **Тізбектер**:
    - Тізбектер жүйелік бағдарламалауда деректерді динамикалық түрде сақтау үшін қолданылады. Мысалы, процессордың жұмыс кезегін, енгізу-шығару операцияларын немесе желідегі пакеттердің тізбегін басқару үшін тізбектер қолданылады.

---

110. Файлдардың барлық типін құру және ашу функцияларын айтыңыз

Файлдардың барлық типтерін құру және ашу функциялары операциялық жүйеде қолданылатын негізгі функциялар болып табылады. Олар файлды ашу, құру және басқару үшін пайдаланылады.

Негізгі функциялар:

1. **Файлды ашу**:
    - **fopen()** — файлды ашуға арналған функция. Ол файлды оқуға, жазуға немесе екі функцияны да орындауға мүмкіндік береді.
    - **open()** — төмен деңгейдегі жүйелік функция, файлды ашуға немесе жаңа файлды құруға мүмкіндік береді.
2. **Файлды жазу**:

- **fprintf()**, **fwrite()** — файлға деректер жазуға арналған функциялар.
- **write()** — төмен деңгейдегі функция, файлға байт түрінде деректерді жазады.

3. **Файлды оқу**:
   - **fscanf()**, **fread()** — файлдан деректерді оқуға арналған функциялар.
   - **read()** — төмен деңгейдегі функция, файлдан деректерді оқиды.
4. **Файлды жабу**:
   - **fclose()** — файлды жабу үшін қолданылады.
   - **close()** — төмен деңгейдегі жүйелік функция, файлды жабу үшін қолданылады.
5. **Файлды жою**:
   - **remove()** — файлды жою үшін қолданылады.

111. Файлдарды оқу және жазу функцияларының прототипі туралы айтып бер

Файлдармен жұмыс істеу кезінде көптеген кітапханалар мен жүйелік функциялар қолданылады. Стандартты енгізу-шығару функциялары файлдармен деректерді оқу және жазу үшін кеңінен пайдаланылады.

Файлдарды оқу және жазу функциялары:

1. **`fopen()`** — Файлды ашу
   - Прототип:
   - `FILE *fopen(const char *filename, const char *mode);`
   - Параметрлер:
     - `filename`: Ашылатын немесе жасалатын файлдың аты.
     - `mode`: Файлды ашу режимі (мысалы, `"r"` — оқуға, `"w"` — жазуға).
2. **`fread()`** — Файлдан деректерді оқу
   - Прототип:
   - `size_t fread(void *ptr, size_t size, size_t count, FILE *stream);`
   - Параметрлер:
     - `ptr`: Оқыған деректерді сақтайтын буфер.
     - `size`: Бір элементтің көлемі.
     - `count`: Оқылатын элементтердің саны.
     - `stream`: Файл ағыны, мысалы, `FILE *` қайтарылған `fopen()` арқылы алынған.
3. **`fwrite()`** — Файлға деректерді жазу
   - Прототип:
   - `size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream);`
   - Параметрлер:
     - `ptr`: Жазылатын деректердің мекенжайы.
     - `size`: Бір элементтің көлемі.
     - `count`: Жазылатын элементтердің саны.
     - `stream`: Файл ағыны.
4. **`fscanf()`** — Форматталған деректерді оқу
   - Прототип:
   - `int fscanf(FILE *stream, const char *format, ...);`
   - Параметрлер:
     - `stream`: Оқу үшін ашылған файл ағыны.
     - `format`: Файлдан оқылатын деректердің форматы.
5. **`fprintf()`** — Форматталған деректерді жазу
   - Прототип:
   - `int fprintf(FILE *stream, const char *format, ...);`
   - Параметрлер:
     - `stream`: Жазылатын файл ағыны.

- `format`: Жазылатын деректердің форматы.

---

112. Файлдарды өшіру, көшіру функцияларының прототипі туралы айтып бер

Файлдармен жұмыс істегенде олардың жойылуы немесе көшіруі де жиі қажет. Бұл операцияларды орындау үшін арнайы функциялар қолданылады.

Файлдарды өшіру және көшіру функциялары:

1. **`remove()`** — Файлды жою
   - Прототип:
   - `int remove(const char *filename);`
   - Параметрлер:
     - `filename`: Жойылатын файлдың аты.
   - Қайтарылатын мән: Файлды сәтті жойған кезде 0 қайтарылады, қате болған жағдайда -1 қайтарылады.
2. **`rename()`** — Файлдың атын өзгерту немесе оны басқа орынға көшіру
   - Прототип:
   - `int rename(const char *oldname, const char *newname);`
   - Параметрлер:
     - `oldname`: Аты өзгертілетін немесе көшіру үшін алынатын файлдың аты.
     - `newname`: Жаңа аты немесе жаңа орын.
   - Қайтарылатын мән: Егер операция сәтті болса, 0 қайтарылады, қате болса -1.

---

113. Каталогтар деген не? Және оны қалай құрамыз?

Каталог — бұл файлдық жүйеде файлдарды немесе басқа каталогтарды ұйымдастыру үшін қолданылатын құрылым. Каталогтар файлдардың сақталуын оңтайландыруға және олардың құрылымын анықтауға көмектеседі.

Каталогты құру:

- Каталогты құру үшін келесі жүйелік функцияларды пайдалануға болады.

1. **`mkdir()`** — Каталогты құру
   - Прототип:
   - `int mkdir(const char *pathname, mode_t mode);`
   - Параметрлер:
     - `pathname`: Құрылатын каталогтың жолы.
     - `mode`: Каталогтың құқықтары. Бұл параметр тек Linux/Unix жүйелерінде қолданылады.
   - Қайтарылатын мән: Егер каталог сәтті құрылса, 0 қайтарылады, қате болған жағдайда -1.
2. **`CreateDirectory()`** (Windows) — Windows жүйесінде каталогты құру
   - Прототип:
   - `BOOL CreateDirectory(LPCSTR lpPathName, LPSECURITY_ATTRIBUTES lpSecurityAttributes);`
   - Параметрлер:
     - `lpPathName`: Құрылатын каталогтың жолы.
     - `lpSecurityAttributes`: Қауіпсіздік сипаттамалары (қосымша параметр).

---

114. Каталогтардың қандай параметрлері бар?

Каталогтар файлдық жүйеде файлдарды ұйымдастыру және оларды басқару үшін бірнеше маңызды параметрлерге ие. Олар файлдар мен каталогтарды дұрыс сақтау мен іздеуге мүмкіндік береді.

Негізгі параметрлер:

1. **Жол (Path)**:
    - Каталогтың нақты орналасуы. Каталогтың толық жолы оның орналасқан жерін анықтайды.
2. **Құқықтар (Permissions)**:
    - Каталогқа кіру, оқу, жазу сияқты әрекеттерді реттейтін құқықтар жиынтығы.
    - UNIX/Linux жүйесінде `r` (оқуға рұқсат), `w` (жазуға рұқсат), `x` (орындауға рұқсат) болып бөлінеді.
3. **Иесі (Owner)**:
    - Каталогқа жауапты пайдаланушы немесе топ.
4. **Уақыт белгісі (Timestamps)**:
    - Каталогтың соңғы өзгертілген уақыты, жасалған уақыты және соңғы кіру уақыты.
5. **Құпиялылық (Security Attributes)**:
    - Қауіпсіздік, яғни каталогтың құқықтары мен қолжетімділік шарттарын белгілейтін параметр.

---

115. `SetFilePointer` функциясының параметрлері туралы айтыңыз

`SetFilePointer()` — бұл функция файлдың ағымдағы көрсеткішін өзгерту үшін қолданылады, яғни файл ішінде деректердің қай жерінен оқу немесе жазу операциясын бастау керек екенін анықтайды.

Прототип:

```
DWORD SetFilePointer(
  HANDLE hFile,                // Файлдың идентификаторы
  LONG  lDistanceToMove,       // Қозғалу қашықтығы
  PLONG lpDistanceToMoveHigh, // Қозғалу қашықтығының жоғары бөлігі
  DWORD dwMoveMethod           // Қозғалу әдісі (негізгі, соңғы және т.б.)
);
```

Параметрлер:

- **`hFile`**: Ашылған файлдың идентификаторы. Бұл идентификаторды `CreateFile()` сияқты функциялар арқылы алуға болады.
- **`lDistanceToMove`**: Файлдың ағымдағы көрсеткішінен қанша орын алға немесе артқа жылжитынын көрсетеді. Мән теріс болған жағдайда көрсеткіш артқа, оң болғанда алға жылжиды.
- **`lpDistanceToMoveHigh`**: Егер жылжудың қашықтығы өте үлкен болса, жоғары бөлігін көрсету үшін қолданылады. Бұл параметрдің мәні көбінесе 0 болады.
- **`dwMoveMethod`**: Жылжытудың негізгі әдісі:
    - `FILE_BEGIN`: Файлдың басынан бастап жылжу.
    - `FILE_CURRENT`: Қазіргі көрсеткіштен бастап жылжу.
    - `FILE_END`: Файлдың соңынан бастап жылжу.

Қайтарылатын мән:

- Сәтті орындалса, файл көрсеткішінің жаңа орнын қайтарады. Қате болған жағдайда `INVALID_SET_FILE_POINTER` мәнін қайтарады, сондықтан қателерді тексеру үшін `GetLastError()` функциясын қолдану қажет.

116. Іздеу дескрипторының параметрлерін айтып бер

Іздеу дескрипторы (Search Descriptor) — бұл файлдарды іздеу үшін қолданылатын параметрлердің жиынтығы. Әдетте, бұл дескрипторлар файлдық жүйелерде файлдарды немесе каталогтарды іздеу үшін арнайы функциялармен қолданылады.

Іздеу дескрипторының параметрлері:

1. **`hFindFile`**:
    - Файлды іздеу операциясын бастағаннан кейін қайтарылатын дескриптор. Бұл дескриптор кейін `FindNextFile()` және `FindClose()` функцияларымен жұмыс істейді.

2. **lpFindFileData**:
    - Бұл құрылым файлдар туралы ақпаратты сақтайды. Мысалы, файл аты, атрибуттары, өлшемі, соңғы өзгерту уақыты және басқа да сипаттамалар.
3. **dwFileAttributes**:
    - Іздеудің қандай файл атрибуттарымен шектелетінін көрсететін параметр. Мысалы, тек каталогтарды, тек жасырын файлдарды іздеу немесе басқа атрибуттарға негізделген іздеу.
4. **FindFirstFile()**:
    - Іздеу операциясын бастау үшін қолданылатын функция, бұл функция іздеуге сәйкес келетін алғашқы файл туралы ақпарат береді. Ол іздеу дескрипторын және файл туралы мәліметтерді қайтарып береді.
    - Прототип:
    - ```
      HANDLE FindFirstFile(
          LPCSTR lpFileName,   // Іздеудің жолы мен файл аттары
          LPWIN32_FIND_DATA lpFindFileData  // Іздеу нәтижелері
      );
      ```

Қолданылатын функциялар:

- **FindFirstFile()**: Іздеуді бастайды.
- **FindNextFile()**: Іздеуден кейін келесі файлды табады.
- **FindClose()**: Іздеуді аяқтайды және ресурстарды босатады.

---

117. Файлдың атрибуттарымен қандай жұмыс жасалады?

Файлдардың атрибуттары — файлдардың қасиеттері мен сипаттамаларын анықтайтын параметрлер болып табылады. Олар файлдарды қалай өңдеу керек екенін көрсетеді.

Файл атрибуттары:

1. **FILE_ATTRIBUTE_READONLY**:
    - Файл тек оқуға арналған, оны өзгертуге болмайды.
2. **FILE_ATTRIBUTE_HIDDEN**:
    - Файл жасырын, яғни әдепкі бойынша көрсетілмейді.
3. **FILE_ATTRIBUTE_SYSTEM**:
    - Файл жүйелік файл болып табылады, оның маңызы бар жүйелік операциялар үшін.
4. **FILE_ATTRIBUTE_DIRECTORY**:
    - Файл каталог екенін көрсетеді.
5. **FILE_ATTRIBUTE_ARCHIVE**:
    - Файл архивтелуге дайын екенін көрсетеді, әдетте резервтік көшірме жасау үшін пайдаланылады.
6. **FILE_ATTRIBUTE_TEMPORARY**:
    - Файл уақытша екенін білдіреді, ол жүйе үшін арнайы қолданылуы мүмкін.
7. **FILE_ATTRIBUTE_NORMAL**:
    - Әдеттегі файл атрибуты, файлға ешқандай ерекше қасиеттер қолданылмайды.

Атрибуттармен жұмыс істеу функциялары:

1. **GetFileAttributes()**:
    - Файл атрибуттарын алу үшін қолданылады.
    - Прототип:
    - ```
      DWORD GetFileAttributes(LPCSTR lpFileName);
      ```
2. **SetFileAttributes()**:
    - Файл атрибуттарын өзгерту үшін қолданылады.
    - Прототип:
    - ```
      BOOL SetFileAttributes(LPCSTR lpFileName, DWORD
      dwFileAttributes);
      ```

      ```

118. Жүйелік бағдарламалаудағы қауіпсіздік мәселелерін атаңыз

Жүйелік бағдарламалау саласында қауіпсіздік мәселелері көптеген аспектілерді қамтиды, себебі жүйелік бағдарламалар мен операциялық жүйелердің дұрыс жұмыс істеуі үшін қауіпсіздік талаптары өте маңызды.

Қауіпсіздік мәселелері:

1. **Буферлік ағымдардың бұзылуы**:
   - Буферді артық жазу немесе артық оқу операциялары бағдарламаның жұмысын бұзып, зиянкестерге мүмкіндік туғызуы мүмкін.
2. **Енгізу деректерін тексермеу**:
   - Кіріс деректерінің дұрыстығын тексермеу бағдарламаның жұмысында қатеге әкелуі мүмкін.
3. **Рұқсатсыз қол жеткізу**:
   - Қолданушылардың рұқсатсыз жүйеге кіруі немесе файлдарға қол жеткізуі.
4. **Кодты орындауға мүмкіндік беретін осалдықтар**:
   - Кодтың орындалуына мүмкіндік беретін осалдықтар, мысалы, командалық жол арқылы жүйені басқару немесе сыртқы кітапханаларды пайдалану арқылы орындалатын код.
5. **Құпия деректерді қорғау**:
   - Құпия деректерді сақтау немесе тасымалдау кезінде қорғау шараларының болмауы.

---

119. Операциялық жүйелерде қауіпсіздік шаралары қалай жүзеге асырылады?

Операциялық жүйелерде қауіпсіздік шаралары әртүрлі деңгейде жүзеге асырылады, соның ішінде пайдаланушыларды аутентификациялау, деректерді қорғау, және жүйелік ресурстарды бақылау.

Қауіпсіздік шаралары:

1. **Аутентификация және авторизация**:
   - Пайдаланушылардың жүйеге кіруін тексеру және олардың рұқсат деңгейін анықтау.
2. **Қол жетімділік құқықтары**:
   - Файлдарға және жүйелік ресурстарға қол жеткізу үшін рұқсат беру және шектеу.
3. **Шифрлау**:
   - Деректерді қорғау үшін шифрлау алгоритмдерін қолдану, әсіресе құпия деректермен жұмыс істегенде.
4. **Қауіпсіздік журналдары**:
   - Операциялық жүйе мен қосымшалардағы қауіпсіздік оқиғаларын жазу және талдау.
5. **Жүйелік қорғаныс құралдары**:
   - Вирусқа қарсы бағдарламалар мен брандмауэрлерді пайдалану.
6. **Қатесіз код жазу**:
   - Кодта қауіпсіздік қатерлерін болдырмау үшін жазылған қауіпсіздікті қамтамасыз ету.

---

120. Жүйе тұрақтылығын қамтамасыз ету үшін қандай әдістер қолданылады?

Жүйенің тұрақтылығын қамтамасыз ету үшін әртүрлі әдістер мен тәсілдер қолданылады, олар бағдарламалық жасақтаманың дұрыс жұмыс істеуі және пайдаланушылар үшін үздіксіз қызмет көрсету үшін өте маңызды.

Тұрақтылықты қамтамасыз ету әдістері:

1. **Қателерді өңдеу (Error Handling)**:

    - Бағдарламада орын алуы мүмкін қателерді алдын ала анықтап, оларды тиісті түрде өңдеу.
2. **Жүйелік мониторинг**:
    - Жүйенің жұмысын үнемі бақылау және ресурстардың тиімді пайдаланылуын қадағалау.
3. **Қайталанатын деректер мен резервтік көшірмелер**:
    - Қорғау мен тұрақтылықты қамтамасыз ету үшін деректердің сақтық көшірмесін жасау.
4. **Ресурстарды басқару**:
    - Жүйелік ресурстардың шектелген және тиімді пайдалануын қамтамасыз ету, мысалы, жад пен процессор уақытын дұрыс бөлу.
5. **Тестілеу және симуляция**:
    - Бағдарламаларды немесе жүйелерді тұрақтылыққа тексеру үшін әртүрлі жағдайларды симуляциялау.
6. **Қолданушы рұқсаттарын бақылау**:
    - Әр түрлі қолданушылардың жүйеде дұрыс және қауіпсіз әрекет етуін қамтамасыз ету арқылы тұрақтылықты қолдау.

121. Жүйелік бағдарламалауда қолданылатын негізгі құралдарды атаңыз

Жүйелік бағдарламалауда қолданылатын құралдар — бағдарламалау процесін жеңілдету және тиімді ету үшін пайдаланылатын арнайы бағдарламалық құралдар. Бұл құралдар жүйе деңгейінде жұмыс істейтін бағдарламалар мен операциялық жүйелерді әзірлеуге арналған.

Негізгі құралдар:

1. **Компиляторлар**:
    - Жүйелік бағдарламалау үшін C, C++ сияқты төмен деңгейлі тілдер үшін компиляторлар пайдаланылады. Мысалы, GCC, Clang.
2. **Дебаггерлер**:
    - Бағдарламаларды ақауларын іздеп табу үшін қолданылатын құралдар. Мысалы, GDB, WinDbg.
3. **Ассемблерлер**:
    - Төмен деңгейдегі тілдерді (мысалы, ассемблер тілі) машиналық кодқа айналдыратын құралдар. Мысалы, NASM, MASM.
4. **Профайлерлер мен трекерлер**:
    - Бағдарлама жұмысының уақытын, ресурстарды қалай тұтынып жатқанын бақылауға арналған құралдар. Мысалы, gprof, Valgrind.
5. **Версиялау жүйелері**:
    - Бағдарламаның дамуын қадағалау үшін, кодтың әртүрлі нұсқаларын сақтау және басқару. Мысалы, Git, SVN.
6. **IDE (Интеграцияланған даму ортасы)**:
    - Бағдарлама жазуға арналған барлық құралдарды бір ортада ұсынатын құралдар. Мысалы, Visual Studio, Eclipse, CLion.
7. **Системалық мониторинг құралдары**:
    - Жүйенің ресурстарын бақылап, жүйелік қатені анықтауға арналған құралдар. Мысалы, top, htop, Process Explorer.

---

122. Дебаггер дегеніміз не және ол жүйелік бағдарламалауда қалай қолданылады?

Дебаггер — бағдарламаның орындалуын қадағалап, қателерді (bug) табу және жөндеуге мүмкіндік беретін арнайы құрал.

Дебаггердің негізгі қызметтері:

1. **Байқау**: Бағдарлама орындалған сайын оның ішіндегі айнымалыларды, жадты және процессордың күйін бақылау.

2. **Қадам-қадам орындату**: Бағдарламаны қадаммен орындауға мүмкіндік береді, бұл қателерді табуға көмек береді.
3. **Нүктелерді белгілеу (Breakpoints)**: Қажетті орындарға тоқтау нүктелерін қою арқылы бағдарламаның орындалуын тоқтатып, оның жағдайын талдауға мүмкіндік береді.
4. **Трассировка**: Бағдарламаның орындалу жолын бақылау, қай жерде және қалай қате пайда болатынын анықтауға көмектеседі.
5. **Қателерді жөндеу**: Қатені түзету үшін айнымалыларды өзгерту немесе бағдарламаның орындалуын басқару.

Жүйелік бағдарламалаудағы рөлі:

- Дебаггерлер жүйелік бағдарламалардың жұмысын дұрыс ұйымдастыру үшін маңызды, өйткені жүйелік бағдарламалар қателерді анықтау және жою кезінде көп қиындық тудырады. Сонымен қатар, олар төмен деңгейдегі ақаулар мен жүйелік ресурстарды басқаруда маңызды құрал болып табылады.

---

123. Жүйелік бағдарламалау үшін қолданылатын профайлер мен трекерлердің маңызы қандай?

Профайлерлер мен трекерлер — бұл бағдарламаларды талдау, олардың өнімділігін арттыру, және ресурстарды тиімді пайдалану мақсатында қолданылатын құралдар.

Профайлерлер:

1. **Өнімділікті талдау**:
   - Профайлер бағдарламаның әрбір функциясының орындау уақытын өлшейді және қай жерлерде көп уақыт кететінін анықтауға мүмкіндік береді.
2. **Ресурстарды бақылау**:
   - Профайлерлер ресурстардың (жад, процессор уақыты, енгізу/шығару) қалай пайдаланылып жатқанын көрсетеді.
3. **Кодтың тиімділігін арттыру**:
   - Бағдарламаның ең баяу немесе ең көп ресурстарды тұтынатын бөліктерін табуға көмектеседі.

Трекерлер:

1. **Қателер мен орындалу жағдайларын қадағалау**:
   - Трекерлер бағдарламаның орындалу барысында әрбір қадамды және деректерді тіркеп отырады.
2. **Жүйелік ресурстарды басқару**:
   - Жүйеде орындалған барлық әрекеттерді бақылап, ресурстарды тиімді пайдалануға көмектеседі.

Маңыздылығы:

- Профайлерлер мен трекерлер жүйелік бағдарламалау кезінде бағдарламаның жұмысын тиімді басқаруға, орындалу жылдамдығын арттыруға, және жүйелік ресурстарды дұрыс бөлуді қамтамасыз етуге мүмкіндік береді.

---

124. Жүйелік бағдарламалау болашағы қандай технологиялармен байланысты болуы мүмкін?

Жүйелік бағдарламалаудың болашағы әртүрлі жаңа технологиялар мен құралдармен тығыз байланысты, бұл аймақтың үнемі даму үстінде екенін көрсетеді.

Болашақ технологиялар:

1. **Жасанды интеллект (AI) және машинамен оқыту**:
   - Жүйелердің өнімділігін арттыру және ресурстарды тиімді пайдалану үшін жасанды интеллект қолданылуы мүмкін. AI жүйелерінің қателерін болжау, жүйенің күйін алдын ала анықтау және тиімді шешімдер қабылдау үшін пайдаланылады.
2. **Микросервистер мен контейнерлер**:

- Жүйелік бағдарламалау контекстінде бұл технологияларды қолдану жүйелердің икемділігін арттырып, қауіпсіздік пен өнімділікті жақсарта алады. Docker және Kubernetes сияқты контейнерлердің танымал болуы контейнерлер мен микросервистердің тиімділігін арттыруға мүмкіндік береді.

3. **Кванттық есептеулер**:
   - Жақын болашақта кванттық есептеу технологиялары жүйелік бағдарламалау мен қауіпсіздік саласында жаңа мүмкіндіктерді аша алады.
4. **Интернет заттары (IoT)**:
   - IoT құрылғыларының кеңеюі жүйелік бағдарламалауды жаңа деңгейге көтеріп, құрылғыларды тиімді басқару мен жүйені тиімді орнатуға мүмкіндік береді.

---

125. Қазіргі уақытта жүйелік бағдарламалау саласында қандай жаңа трендтер байқалады?

Қазіргі уақытта жүйелік бағдарламалау саласында бірнеше жаңа трендтер мен өзгерістер байқалады, бұл саланың дамуы мен жетілдірілуін көрсетеді.

Жаңа трендтер:

1. **Автоматизация және DevOps**:
   - Жүйелік бағдарламалау саласында DevOps мәдениетінің кеңінен таралуы бағдарламаны дамыту мен оның операциялық процестерін автоматтандыруды білдіреді. Жүйелік бағдарламалаушының міндеті тек қана код жазумен шектелмей, сонымен қатар жүйенің үздіксіз интеграциясы мен жеткізілуін қамтамасыз ету.
2. **Құрылғының деректерді өңдеуін оңтайландыру**:
   - Параллель есептеулер, көп ағынды және көп ядролы өңдеу жүйелері арқылы өнімділікті арттыру. Мұның көмегімен жүйелер жылдам жұмыс істеп, деректерді өңдеу көлемі арттырылады.
3. **Қауіпсіздік аспектілері**:
   - Киберқұқық бұзушылықтар мен қауіпсіздікке қатысты мәселелердің көбейіп келе жатқанын ескерсек, жүйелік бағдарламалауда қауіпсіздік аспектілеріне ерекше назар аударылады. Қазіргі уақытта қауіпсіздікті бірінші орынға қою — жаңа тренд.
4. **Жүйелерді виртуализациялау және контейнерлеу**:
   - Виртуализация және контейнерлеу технологиялары серверлік ресурстарды тиімді пайдалану және жүйелік бағдарламалаудың икемділігін арттыру үшін кеңінен қолданылады.
5. **Жоғары өнімділікті есептеулер**:
   - Жүйелік бағдарламалау аясында жоғары өнімділікті есептеулердің, мысалы, деректерді талдау мен ғылыми есептеулердің маңызы артуда.