1. **Tytuł projektu: BJOS** - Windowsopodobny system operacyjny
2. **Język programowania:** Java
3. **Wykonawcy :**
    1) Piotr Kaszuba - System plików
    2) Mikołaj Fedec - Pamięć operacyjna
    3) Sławomir Asimowicz - Interpreter programów assemblerowych
    4) Mariusz Hajer - Shell + programy testowe
    5) Filip Krzemień - Komunikacja międzyprocesowa
    6) Szymon Chojnowski - Mechanizmy Synchronizacyjne (Semafory i zamki)
    7) Karol Czub - Planista/Zarządzanie Procesorem
    8) Dominik Krystkowiak - Zarządca procesów

4. Deklaracje podstawowych struktur danych

Planista/Procesor:

```
Queue <PCB> [] TabKolProc=new LinkedList[16];
```

System plików:

```
this.disk = new byte[this.sector_number*this.sector_size];
```

Zarządzanie pamięcią operacyjną i pamięcią wirtualną:

```
char [] memory = new char[frames_number*page_size];
```

Zarządzanie procesami:

```
static LinkedList<PCB> lista_PCB;
```

Komunikacja międzyprocesowa:

```
MsgQueue msgqueue = new MsgQueue();
```

Mechanizmy synchronizacyjne:

```
private LinkedList<PCB> LockQueue;
private LinkedList<PCB> SemaphoreQueue;
```

Interpreter:

```
public class Interpreter
{
	private int RejestrA;
	private int RejestrB;
	private int RejestrC;
	private int licznik_rozkazow;
	private int adres_pamieci;
}
```

5. Nagłówki głównych procedur/funkcji realizujących wybrany mechanizm/algorytm

Planista/Procesor:

public void DodajLiczniki() //zwiększa licznik procesu, który jest obecnie w running oraz liczniki bezczynności pozostałych procesów co każdą wykonaną linię programu.

public void UstawPriorytety() //Zwiększa/zmniejsza priorytety dynamiczne, w zależności od liczników bezczynności, tak aby zapobiec zagłodzeniu procesów o niskim priorytecie (w tym wypadku oczekiwanie dłuższe niż 8 linii, czyli dwa kwanty czasu, powodowało wzrost priorytetu dynamicznego danego procesu do maksymalnego - 15 i powrót do priorytetu bazowego, po wykorzystaniu swojego kwantu czasu (4 linie).

public void UstawNastepny() throws InterruptedException //Ustawia kolejny proces do running, w zależności od ich gotowości oraz priorytetów dynamicznych. Jeżeli żaden proces o tym samym lub wyższym priorytecie nie jest gotowy, a proces który do tej pory był w running jest gotowy i jeszcze się nie zakończył - przyznaje mu kolejny kwant czasu.

System plików:

//pozwala na wyedytowanie całego wpisu katalogowego o kolejnym numerze entrynr, wypełniając jego odpowiednie pola resztą parametrów
1) private boolean complex_entry_edit(int entrynr, int att, byte[]name, int first_jap, int rozmiar)

//odczytuje z tablicy FAT numer kolejnej JAP w polu JAP_number
2) private int FAT_next_JAP(int JAP_number)

//wpisuje do tablicy FAT numer kolejnej JAP (next_JAP) w pole JAP_number
3) private void FAT_chain(int JAP_number, int next_JAP)

//dopisuje na końcu pliku “name” ciąg utworzony z “data”, jeśli potrzeba powiększa łańcuch //JAP, edytuje wpis katalogowy(rozmiar+size), zwraca fałsz w przypadku błędu przy zapisie
4) private boolean save_file(byte[] name, byte[] data, int size)

Zarządzanie pamięcią operacyjną i pamięcią wirtualną:

//zapisuje plik w pamięci wirtualnej
public void virtual_memory(int procesID, String Sname);

//wysyła ramkę do pamięci operacyjnej z pamięci wirtualnej
public char [] send_frame(int frame_number);

//zwraca znak do interpretera
public char read_memory(int virtualAddress, int procesID);

Mechanizmy synchronizacyjne:

public void waits(PCB Process)
Dekrementuje zmienną semaforową, dodaje proces do kolejki i wstrzymuje go jeśli coś już go blokuje.

public void signal()
Podnosi semafor, jeśli jakiś proces oczekuje w kolejce usuwa go z niej i zmienia jego stan na gotowy.

public void lock(int PID)
Blokuje zamek, zapisuje ID procesu blokującego, dodaje też proces do kolejki i wstrzymuje go jeśli zamek jest zablokowany.

public void unlock(int PID)
Zwalnia zamek - tylko wtedy gdy ID procesu równa się procesowi który blokuje zamek. Jeśli jakiś proces oczekuje w kolejce usuwa go z niej i zmienia jego stan na gotowy.

Zarządzanie procesami:

int nowy_proces(String naz, int prio) – funkcja wywołuje konstruktor klasy PCB, dodaje utworzony proces do listy PCB, wywołuje jedną z metod zarządzania procesorem (dodającą proces do tablicy kolejek), zwraca id utworzonego procesu.

void usun_proces(int identyfikator) – usuwa proces z listy PCB poprzez podany identyfikator korzystając z iteratora oraz wywołuje metodę zarządzania procesorem (usuwającą proces z tablicy kolejek).

PCB znajdz_proces(int identyfikator) – wyszukuje proces w liście PCB po identyfikatorze oraz zwraca ten proces. Jeżeli proces nie zostanie znaleziony zwraca null.

Komunikacja międzyprocesowa:

public static void wyslijKomunikat(int PID,int od_kogo, String komunikat) – funkcja wpisuje komunikat od procesu i zapisuje parę wartości (PID nadawcy oraz komunikat) do kolejki komunikatów procesu o ID od_kogo.

public static String odbierzKomunikat(int PID) – funkcja próbuje odczytać komunikat z kolejki komunikatów zadanego procesu. W przypadku, gdy kolejka jest pusta następuje zawieszenie procesu pod semaforem do momentu dodania komunikatu.

public static void wyswietlKolejke(int PID) – funkcja wyświetla aktualną kolejkę komunikatów dla danego procesu.

Interpreter:

//metoda która pobiera rejestry z PCB
private void PobierzRejestry(PCB rozkaz)

//metoda która wysyła rejestry do PCB
private void WyslijRejestry(PCB rozkaz)

//metoda która pobiera z pamięci znak na podstawie adresu i id procesu
private String Zczytajzpamieci(int adres, int idprocesu)

Shell:

//metoda, które poprzez odpowiednie wywołania tworzy programy zapisane w plikach
private void Install_programs();

//metoda, która obsługuje polecenia użytkownika dawane systemowi
private void Shell_loop();

6. Opis interfejsu niezbędnego do współpracy z innymi zadaniami realizowanymi w grupie projektowej

Planista/Procesor:
procedury użyte do współpracy z innymi podzespołami:

1) **public void Dzialaj(Interpreter1 slawek) throws InterruptedException** //po ustawieniu priorytetów i runninga, wywołuje interpreter by ten zczytał linię i dodaje liczniki.
2) **public void DodajProces(PCB proces)** //procedura wywoływana przez zarządce procesów, która dodaje dany proces od razu do kolejki procesów oczekujących na procesor. Wywoływana przy dodawaniu nowego procesu.
3) **public void UsunProces(int iden)** //procedura wywoływana przez zarządce procesów, która usuwa dany proces z kolejki procesów oczekujących na procesor. Wywoływana przy zakończeniu danego procesu.
4) **public void WypiszGotowe()** //procedura do komunikacji z shellem. Na życzenie użytkownika wypisuje ona wszystkie procesy w tablicy kolejek, łącznie z ich stanem i priorytetem.

System plików:
Interfejs do współpracy tworzą funkcje:

1) wykorzystywane przez użytkownika w celu modyfikowania zawartości dysku i przechowywania programów:
   a) **private boolean save_file(byte[] name, byte[] data, int size)** - zapis pliku na dysku
   b) **public boolean write_to_file(String Sname, String Stext)** - pisanie do pliku
   c) **public boolean rename_file(String Sname, String newSname)** -zmień nazwę pliku
   d) **public boolean delete_file(String Sname)** - usunięcie pliku

2) wykorzystywane w mechanizmie systemu - wysyłania pliku przetworzonego do tablicy bajtów wraz z podanym rozmiarem do pamięci.

Odczytanie/załadowanie pliku/programu do pamięci RAM :
public byte[] open_file(String Sname)

Zarządzanie pamięcią operacyjną i pamięcią wirtualną:
Interfejs do współpracy tworzy funkcja wysyłająca znak do interpretera
oraz funkcje wyświetlające zawartość pamięci operacyjnej, pamięci wirtualnej oraz listy wolnych ramek.

public void print_memory();

public void print_virtual_memory();
public void print_list();

Mechanizmy synchronizacyjne:

Wywołuje procedure zarządcy procesów do zmianu stanu procesu przebywającego lub próbującego się dostać do semafora/zamka.
Procedury: **Proces_Gotowy()**, **Proces_Wstrzymany().**

Shell może podejrzeć kolejkę na semaforze przy pomocy metody:
public void show_semaphore_queue()

Zarządzanie procesami:

Klasa PCB zawiera pola typu **PageTable**, **MsqQueue**, semaphore do których dostęp ma odpowiedni moduł (zarządzanie pamięcią, komunikacja międzyprocesowa oraz mechanizmy synchronizacyjne).

Funkcja PCB **znajdz_proces(int identyfikator)** wykorzystywana jest przez moduły do wyszukiwania procesu w liście.

Dwie funkcje testujące:
void lista_procesow() - wypisuje wszystkie procesy znajdujące się w liście
void wypisz_proces(int id) – wypisuje parametry zadanego procesu (identyfikator, stan, priorytet, stan rejestrów oraz licznika rozkazów)

Komunikacja międzyprocesowa:

Współdziała z zarządcą procesów oraz systemem plików.
Zarządca procesów pozwala na dostęp do kolejki komunikatów umieszczonej w PCB danego procesu. Korzysta z funkcji **zwroc_proces** klasy **Zarzadzanie_procesami**.
System plików pozwala na zapisywanie odczytanego komunikatu na dysk. Korzysta z funkcji **create_file**, **write_to_file**, **close_file** klasy **FAT8**.

Z klasą IPC można współpracować poprzez używanie funkcji pozwalających na dodawanie/odczytywanie komunikatów:
public static void wyslijKomunikat(int PID,int od_kogo, String komunikat),
public static String odbierzKomunikat(int PID),
public static void wyswietlKolejke(int PID).

Interpreter:

public void WykonajRozkaz(PCB id)
Metoda wywoływana przez planistę która na podstawie ID procesu wykonuje pobranie rejestrów i licznika rozkazów z PCB, pobranie z pamięci pojedynczego rozkazu i wykonanie go poprzez interpretacje i wywołanie odpowiedniej funkcji z poszczególnych modułów lub wykonywanie poszczególnych operacji na rejestrach. Na końcu wysyłany jest stan rejestrów i licznika rozkazów do PCB danego procesu.

Shell:
Funkcja, która pełni rolę pętli głównej wywołuje wszystkie udostępnione przez inne moduły metody. W pierwszej kolejności pobiera ona od użytkownika komendę, a następnie uruchamia odpowiednie procedury/funkcje:
private void Shell_loop();

7. Uwagi dodatkowe