1. **Tytuł projektu:** bigOS → system operacyjny grupy L5b
2. **Język programowania:** Java
3. **Wykonawcy** 
   1. Wojciech Kulczak → Zabezpieczenia ->583 linie kodu
   2. Anna Lehnhardt → Interpreter → 890 linii kodu
   3. Konrad Kęciński → Interfejs → 597 linii kodu
   4. Jonasz Łapiński → Zarządzanie procesorem metoda priorytetową → 274 linii kodu
   5. Maciej Sobkowiak → Zarządzanie pamięcią operacyjną metodą przydziału obszarów ciągłych przesuwalnych → 220 linii kodu
   6. Piotr Kwiatkowski → Zarządzanie procesami
   7. Rafał Ewiak → Mechanizmy synchronizacyjne: semafory całkowitoliczbowe → 70 linii kodu
   8. Weronika Kowalska → Zarządzanie plikami za pomocą tablic FAT → 568 linii kodu
   9. Wojciech Lulek → Komunikacja międzyprocesowa za pomocą komunikatów tekstowych
4. **Nazwy i przeznaczenie podstawowych struktur danych w danym module** (min. 1, maks. 3)
   1. List<User> UL->lista użytkowników  
      Table<SaID,String,Integer> ACM-> macierz dostępu do plików.
   2. String calyRozkaz → przechowuje aktualnie wykonywany rozkaz assemblerowy
   3. String user → przechowuje nazwe aktualnie zalogowanego użytkownika ArrayList<String> parts → lista przechowująca poszczególne części komendy wprowadzonej przez użytkownika
   4. List<Queue<Process>> *queuesPCB*; → lista kolejek będących w stanie Ready;

Process dummy; → zmienna przechowująca proces Dummy o priorytecie 0;

Process running; → zmienna przechowująca aktualnie wykonywany proces;

* 1. char[] memory → jednowymiarowa tablica imitująca RAM o wielkości 256B  
     Map<Integer,Integer> allocatedPartitions → mapa zajętych obszarów   
     Map<Integer,Integer> freePartitions → mapa wolnych obszarów
  2. Map<Integer, Process> processes → przechowywanie orginałów PCB; Vector<String> taken\_names → przechowywanie zajętych nazw w celu zabezpieczenia przed powstaniem kilku procesów o tej samej nazwie; Map<String, Integer> names → mapa, w której kluczem jest nazwa procesu, a wartością jego PID, ułatwia odwoływanie się do konkretnych procesów;
  3. Queue<Integer> process\_queue; → kolejka zawierająca PID procesów w stanie Waiting

Int value ; → zmienna semaforowa oznaczająca ilość dostępnych zasobów, a gdy jest ujemna liczbę procesów w kolejce

* 1. Integer[] FAT – tablica FAT; char[] data – dysk posiadający 32 bloki po 32 bajty; List<File> root – lista zawierająca pliki, katalog główny,
  2. Queue<Message> messages\_queue - kolejka wiadomości otrzymanych przez dany proces

1. **Nagłówki (w języku C++ lub Java) podstawowych procedur/funkcji realizujących dany moduł** (min. 1, maks. 3)
   1. public boolean add2ACM(String name, SaID a);  
      public boolean check\_permission(String input, SaID c\_user, String filename);  
      public boolean create\_acc(String name, String passwd);
   2. private static void getOrder();

public static void go(int how\_many);

private static void start(String filename);

* 1. public void system();  
     public void cut();   
     public void execute();
  2. public static void schedule(); public static void makeOlder();   
     public static void add(Process toAdd);
  3. public static void writeMemory(String fileName,Integer PID);  
     public static void removeProgram(int PID);  
     public static void move();
  4. public static void create\_process(String name, String file\_name, int priority);   
     public static void delete(int PID) oraz public static void delete(String name);   
     public static void add\_to\_CPU(int PID),
  5. public void wait\_s(int pid);

public void signal\_s();

public void print\_queue();

* 1. public static boolean create(String name, String user);   
     public static boolean write(String name, String data);   
     public static String read(String fileName, int from, int howMany);
  2. public boolean send\_message(int receiverPID, String text);   
     public boolean send\_message(int receiverPID, int size, int addres);   
     public boolean read\_message(int addres)

1. **Jednozdaniowe uwagi dotyczące interfejsów do pracy krokowej i współdziałania z innymi modułami** (maks. 3 w danym module)
   1. Każda z operacji na plikach kontorlowana jest przez moduł bezpieczeństwa- sprawdzane są uprawanienia użytkownika każdego z plików.Współpraca ze środowiskiem przy zarządzaniu kontami użytkowników. Komunikaty do każdych z operacji.
   2. Możliwość wykonania po kolei każdego rozkazu, po którym wyświetlane są rejestry aktualnego procesu. Współdziałanie z procesami, żeby pobierać i uaktualniać PCB. W programach wywoływane są funkcje klas zarządzających plikami oraz wysyłaniem wiadomości międzyprocesowych.
   3. Po wpisaniu komendy przez użytkownika, interface informuje użytkownika o funkcjach których wywołanie spowodowała dana komenda. Z poziomu interfejsu można komunikować się z pozostałymi modułami.
   4. Możliwość wyświetlenia PCB aktualnie wykonywanego procesu, a także zawartości kolejek przechowujących procesy w stanie Ready, ponadto moduł opisuje to co robi w konsoli systemowej.
   5. Możliwość wypisania zawartości pamięci RAM w trzech wariantach (jeden znak, przedział, cała pamięć).  
      Komunikaty dotyczące działania funkcji (dodawanie, usuwanie, przesuwanie).  
      Wykorzystanie semaforów.
   6. Umożliwiam wyświetlenie PCB wszystkich oraz konkretnych procesów, dodatkowo metody działające w ramach funkcjonalności modułu opisują swoje działania w konsoli w trakcie wykonywania.
   7. Wyświetlenie koleji procesów w stanie Waiting po dodaniu procesu do tej kolejki, oraz procesów wchodzących i wychodzących z kolejki. Semafory są zaimplementowane w modulach: RAM, komunikacja międzyprocesowa i system plików.
   8. Możliwość wyświetlenia bieżącej zawartości dysku, wektora bitowego, tablicy FAT oraz FCB pliku wraz ze stanem semafora dla danego pliku.
   9. Możliwość wyświetlenia zawartości kolejki wiadomości otrzymanych od innych procesów oraz stanu semafora regulującego odczytywanie wiadomości.