S1) Algorytm Cristiana++|
Celem projektu jest implementacja algorytmu synchronizacji zegarów z pasywnym serwerem czasu.
Pierwszym etapem jest stworzenie programu symulującego działanie niedoskonałego zegara (z dryfem). Osobny wątek procesu powinien co wypisywać aktualny czas raz na symulowaną sekundę. Następnie na bazie tego programu można przystąpić do implementacji serwera i klientów.
Serwer zna dokładne wskazanie czasu, klient raz na jakiś czas odpytuje serwer(y) o czas i przestawia swój zegar. Serwer powinien symulować opóźnienie sieci (przy obsłudze żądania od klienta można np. użyć `Thread.sleep`). Klient powinien potrafić odpytać jeden serwer wielokrotnie lub wiele serwerów na raz i wybrać najlepszą odpowiedź.

S2) Serwer nazewniczy+|
Celem projektu jest stworzenie serwera nazewniczego, który umożliwi klientom rejestrowanie swoich punktów końcowych. Każdy wpis w tabeli nazewniczej powinien mieć ustalone TTL, po której wpis zostanie usunięty (o ile klient nie przedłuży ważności nazwy)

S3) Serwer heartbeat+|
Celem projektu jest stworzenie serwera monitorującego stan/dostępność klientów. Serwer na starcie dysponuje listą klientów których powinien monitorować. Klienci mogą włączać się i wyłączać w dowolnym momencie, serwer powinien logować informację o podłączeniu się/odłączeniu klienta.

S4) Dostęp do sekcji krytycznej - algorytm pierścieniowy+|
Celem projektu jest stworzenie z kilku procesów logicznego pierścienia. Uruchamiając proces podajemy punkt końcowy następcy. Następnie, po utworzeniu pierścienia procesy powinny przekazywać sobie żeton, który uprawnia do wejścia do sekcji krytycznej. Procesy w sposób solowy wchodzą do sekcji krytycznej i spędzają w niej losowy przedział czasu. Sekcją krytyczną jest proces, który nasłuchuje na dobrze wszystkim znanym punkcie końcowym.

M1) Algorytm z Berkley+|
Pierwszym etapem jest stworzenie programu symulującego działanie niedoskonałego zegara (z dryfem). Osobny wątek procesu powinien co wypisywać aktualny czas raz na symulowaną sekundę. Następnie na bazie tego programu można przystąpić do implementacji serwera i klientów.

1. Serwer rozpoczyna nasłuchiwanie na znanym wszystkim punkcie końcowym
2. Klienci rejestrują się w serwerze
3. Serwer raz na jakiś czas synchronizuje zegary (swój i klientów)

M2) Rozproszony serwer nazewniczy|
Celem projektu jest stworzenie grupy serwerów nazewniczych. Klient może skontaktować się z dowolnym serwerem nazewniczym w celu dodania lub wyszukania określonej nazwy. Serwer może działać w następujący sposób:

1. grupa serwerów może podejmować starania o utrzymanie tabel nazewniczych w stanie spójnym (serwery wymieniają się informacjami o aktualizacjach wpisów w tabeli)
2. jeżeli serwer nie zna nazwy o którą pyta klient, to wysyła zapytanie do innych serwerów (serwery musza o sobie nawzajem wiedzieć)

Pamiętaj, że dowolny serwer może zostać w dowolnym momencie wyłączony/opuścić grupę

M3) Algorytm Tyrana+|

Pierwszy proces tworzy grupę, który zostaje koordynatorem. Każdy kolejny proces możne dołączyć do jednej z grup. W grupie jest jeden koordynator, który może wyłączyć się/ulec awarii. Gdy członek grupy zauważy nieaktywność koordynatora, to rozpoczyna elekcję mającą na celu wyłonienie nowego koordynatora w grupie

M4) Pchane i ciągnięte kopie pamięci|

Mamy 3 rodzaje procesów:

1. Pamięć Trwała (PT) jest procesem w którym mamy możliwość edycji w tybie interaktywnym tablicy haszującej (klucz:wartość). Pamięć trwała wypycha zmiany w tablicy do Kopii Pamięci (KP)
2. Kopa Pamięci - posiada kopię tablicy haszującej, aktualizacje tablicy otrzymuje od PT
3. Klienta (K) - na staricie podłącza się do jednej z KP (podając na starcie punkt końcowy KP). Klient może odpytywać KP o wartości w tablicy. K posiada również pamięć podręczną. Jeżeli klucz o który pyta K jest w pamięci podręcznej to K pobiera tą wartość, jeżeli nie ma w pamięcie podręcznej lub jest przeterminowana to K pyta

---

L1) BeeTorrent|

Projekt ma na celu stworzenie sieci P2P działającej na podobnej zasadzie to BitTorrent i umożliwiającej wymianę plików. W komunikacji biorą udział 2 typy procesów: peer i tracker. Trackery przechowują informację o tym jakie peery są aktywne, i jakie pliki posiadają. Peer może dodawać pliki do sieci lub je pobierać. Gdy peer chce pobrać jakiś plik to najpierw musi skontaktować się z trackerem w celu ustalenia jacy klienci posiadają ten plik. Następnie powinien poprosić każdego klienta o wysłanie jakiegoś fragmentu pliku (e.w. całości).

L2) Rozproszona tablica haszująca|

Celem projektu jest stworzenie rozproszonej tablicy haszującej, przechowującej wartości w postaci klucz:wartość. Tablica przechowywana jest na serwerach w taki sposób, że każdy serwer posiada część kluczy. Klient dodaje parę (klucz,wartość) do tablicy a zadaniem serwera/serwerów jest uzgodnienie na którym serwerze klucz faktycznie zostanie dodany. Podobnie, gdy klient połączy się z serwerem który fizycznie nie posiada klucza w pamięci, to serwer powinien podjąć starania o odnalezienie żądanego klucza na innych serwerach. Metody używane przez klienta: value = get(key), set(key, value), key_exists(key)

L3) Kupcy|

W krainie Foobar żyją kupcy, którzy nie lubią bez potrzeby podróżować. Handlują oni następującymi towarami: rybami, chlebem i jabłkami. Są dwa rodzaje kupców: sprzedający i kupujący. Ponieważ kupcy nie lubią podróżować, to do zawierania transakcji używają gołębi pocztowych. Kupujący, który chce kupić jakiś produkt wysyła gołębia pocztowego do jednego (np. losowego) ze swoich sąsiadów, a ten jeżeli posiada taki towar na sprzedaż to wysyła stosowną wiadomość do kupującego i dochodzi to transakcji. W przypadku, gdy sąsiad nie posiada towaru, którego szuka kupujący, to przesyła gołębia kupującego dalej, do swojego innego losowego sąsiada tak długo, dopóki gołąb znajdzie sprzedającego lub padnie z wycieńczenia (wszystkie gołębie mają określoną maksymalną liczbę przeskoków). Gdy sprzedający zostaje w końcu znaleziony, to odsyła on gołębia pocztowego w drogę powrotną a ten, wracając po swoich śladach, dociera do kupującego i dopiero wtedy dochodzi do bezpośredniej komunikacji i transakcji między kupcami.

Technicznie rzecz biorąc, kupcy tworzą sieć Peer-to-peer. Przyjmij, że jest N kupców (hostów) i każdy kupiec zna wszystkich innych (punkty końcowe wszystkich hostów mogą być zapisane np. w pliku konfiguracyjnym, identycznym dla każdego hosta).

Każdy kupiec (host) przyjmuje losową rolę sprzedawcy lub kupującego. Sprzedawca na początku nie posiada żadnego towaru, a gdy w dowolnym momencie zabraknie mu towaru to uzupełnia zapasy - otrzymuje losową ilość losowo wybranego towaru. Kupujący wybiera losowo towar, który chce kupić. Co jakiś czas, lub po sfinalizowaniu transakcji, kupujący zaczyna poszukiwania kolejnego losowo wybranego towaru.

Każdy host w sieci P2P powinien implementować następujące funkcje:
- szukaj(nazwa_towaru, ilość_przeskoków) - ta funkcja powinna przeszukiwać sieć w poszukiwaniu towaru. Sprzedawca posiadający towar powinien odpowiedzieć na taką wiadomość przesyłając swoje ID. Każdy inny kupiec przekaże komunikat dalej zmniejszając ilość_przeskoków o 1, chyba że wartość ta osiągnie 0 - wtedy należy zaprzestać przekazywania.
- odpowedz(idSprzedawcy) - ta funkcja powinna przesłać odpowiedź od nadawcy do kupującego.
- kup(idSprzedawcy) - przy pomocy tej funkcji, kupujący powinien się skontaktować bezpośrednio ze sprzedawcą. Zakup powinien zmiejszyć o 1 ilość jednostek towaru u sprzedającego.