PSI Zadanie 24Z

Początek realizacji: 04.12.2024 Projekt wstępny do: 14.12.2024; **Koniec realizacji: 24.01.2025**
punktacja: 0 - 40 p. (zaliczenie od 20 p.); **Zespoły:** 4 os

Treść: Napisać program obsługujący prosty protokół P2P (Peer-to-Peer). Założenia:

- Zasób to obiekt z danymi binarnymi identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach.
- Rozmiar zasobu jest znaczny (tj. większy od jednorazowego transferu sieciowego)
- Początkowo dany zasób znajduje się w hoście, następnie może być propagowany do innych hostów w ramach inicjowanego przez użytkownika "ręcznie" transferu (patrz dalej) – raz pobrany zasób zostaje zachowany jako kopia.
- Tak więc, po pewnym czasie działania systemu ten sam zasób może znajdować się w kilku hostach sieci.
- Program ma informować o posiadanych lokalnie (tj. w danym węźle) zasobach i umożliwiać ich pobranie.
- Program powinien umożliwiać:
  - wprowadzanie przez użytkownika nowych zasobów – z lokalnego systemu plików,
  - pobieranie konkretnych nazwanych zasobów ze zdalnego hosta (jednego na raz)
  - rozgłaszanie informacji o posiadanych lokalnie zasobach.
- W przypadku pobierania zdalnego zasobu użytkownik decyduje, skąd zostanie on pobrany.
- Zasób pobrany do lokalnego hosta jest kopią oryginału, kopia jest traktowana tak samo jak oryginał (są nierozróżnialne) – t.j.: istnienie kopii jest rozgłaszane, tak samo jak oryginału.
- Należy zwrócić uwagę na różne obsługę różnych sytuacji wyjątkowych – np. przerwanie transmisji spowodowane błędem sieciowym.
- Lokalizacja zasobów ma następować poprzez rozgłaszanie – wskazówka: użyć prot. UDP, ustawić opcje gniazda SO_BROADCAST, wykorzystać adresy IP rozgłaszające (same bity "1" w części hosta).
- Interfejs użytkownika – wystarczy prosty interfejs tekstowy, powinien on jednak obsługiwać współbieżny transfer zasobów (tj. Nie powinien się blokować w oczekiwaniu na przesłanie danego zasobu)

Warianty funkcjonalne:

- **W11** – Wprowadzić dodatkową funkcję kasowania zasobu, kasowanie powoduje usunięcie wszystkich kopii (wykorzystać rozgłaszanie);
- **W12** – zasymulować błędy protokołu takie jak: zgubienie datagramu UDP i zerwanie sesji TCP; przetestować reakcję programu na w.w. błędy; Błędy powinny być generowane przez dodatkowy moduł podlegający parametryzacji (częstość i charakter błędu).
- **W13** – całość komunikacji (przesyłania zasobu) zrealizować na UDP, dodatkowo dla uproszczenia można przyjąć, że zasób mieści się w całości w jednym datagramie (datagram danych może być zgubiony – należy to uwzględnić)

Warianty implementacyjne:

- **W21** – implementacja w Pythonie
- **W22** - implementacja w C/C++

Uwaga: należy starannie zaprojektować protokół. Już w sprawozdaniu wstępnym należy szczegółowo go opisać, co oznacza podanie formatów wszystkich komunikatów (można je podać jako struktury danych) oraz diagramy czasowe / diagramy przejść dla komunikacji!

Zobacz instrukcję realizacji na kolejnej kartce!

Instrukcje dot. realizacji projektu:

Co powinien zawierać projekt wstępny (c.a. 4 strony)

Sprawozdanie musi być dostarczone mailem w formacie pdf. Ocena ze sprawozdania wstępnego zostanie wystawiona w ciągu 5-7 dni roboczych od dostarczenia, w przypadku oceny < 8 p. konieczne jest omówienie i poprawienie sprawozdania. Konsultowanie się przed przekazaniem sprawozdania nie jest jest wymagane, ale w razie jakichkolwiek wątpliwości zapraszam.

Sprawozdanie **wstępne** powinno zawierać:

1. temat zadania, treść zadania, skład zespołu z wyróżnionym kontaktem lidera, datę przekazania
2. interpretację treści zadania (doprecyzowanie) – rozpisanie funkcji
3. krótki opis funkcjonalny – "black-box", najlepiej w punktach
4. opis i analizę poprawności stosowanych **protokołów komunikacyjnych** (wskazane - z rysunkami, np. zależnosci czasowych przy wymianie komunikatów, oraz postać/formaty komunikatów – np. w postaci tabelek lub rozpisanych w C struktur/obiektów)
5. planowany podział na moduły i strukturę komunikacji między nimi (być może z rysunkiem)
6. zarys koncepcji implementacji (język, biblioteki, narzędzia, etc.)

Nie należy opisywać kwestii znanych i omawianych na wykładzie, np. zasady funkcjonowania API gniazd, funkcji systemowych, standardowych narzędzi programistycznych, itp.

Co powinien zawierać projekt ostateczny (około 8 stron)

1. To co projekt wstępny (copy&paste + ewentualne zmiany i rozszerzenia)
2. Opis najważniejszych rozwiązań funkcjonalnych wraz z uzasadnieniem (np opis: struktur danych, kluczowych funkcji, itp.)
3. Opis interfejsu użytkownika
4. Postać wszystkich plików konfiguracyjnych, logów, itp.
5. Wykaz wykorzystanych narzędzi, itp.
6. **Opis testów i wyników testowania**

Uwaga:

- **Kodowanie**: język C/C++ lub Python (wg. wariantu), środowisku Linux (lub inny Unix: MacOS, BSD, ...)
- **Pokaz** powinien odbywać się w środowisku obejmującym co najmniej 3 połączone siecią "komputery", np. przez VPN, dozwolone jest środowisko zwirtualizowane, preferowany jest serwer kontenerowy bigubu.
- **(e)Konsultacje** w trakcie realizacji projektu nie wymagają obecności całego zespołu.
- **Sprawozdanie końcowe i pokaz** funkcjonowania **musi** odbywać się w obecności całego zespołu, pozostałe warunki formalne – patrz spr. wstępne.
- B. ważne jest precyzyjne opisanie obsługi **sytuacji wyjątkowych i reakcji na błędy**
- B. ważne jest szczegółowe opisanie przeprowadzonych testów – UWAGA: testy nie mają na celu wykazania, że program **działa** poprawnie. Test ma na celu wykazanie, że program **nie działa** poprawnie!
- Komunikacja e-mail – projekt wstępny, końcowy i kod źródłowy prosze wysyłać na adres: **grzegorz.blinowski@pw.edu.pl** , w temacie wiadomości należy wpisać: "[PSI2023L] nazwisko-lidera sprawozdanie wstępne/końcowe/źródla"
- Kod źródłowy proszę przesyłać **wyłącznie** w formatach: **.zip, .tgz lub tar.gz** na adres j.w., **dopiero po** zaaprobowaniu programu i sprawozdania końcowego.
- Spakowany kod źródłowy **nie może zawierać plików binarnych** (.o, .a, wykonywalnych, etc.),
- **Punktacja**: proj. wstępny: 10 p.; ogólna ocena realizacji projektu: 10 p.; sprawozdanie końcowe: jakość i kompletność: 10 p, jakość kodu z punktu widzenia inżynierii oprogramowania (właściwe wykorzystanie dostępnych funkcji i mechanizmów systemowych, komentarze, czytelny podzial funkcjonalny i na moduły, nazewnictwo funkcji i zmiennych, poprawne skonstruowanie plików nagłówkowych (.h), logowanie, itd.): 10 p.; w sumie: 40 p.