Zadání semestrální práce z předmětu KIV/UPS

- síťová hra pro více hráčů, architektura server-klient (1:N), PC
- server: C/C++
- klient: Java, C# (i např. Unity), Kotlin nebo jiný vysokoúrovňový jazyk (musí schválit cvičící)

Varianty zadání:

- tahová hra (prší, mariáš, šachy, piškvorky, ...)
- real-time hra (různé skákačky a střílečky, tanky, "Bulánci", ...)
- pseudo-real-time (Pong, Arkanoid, ...)

Požadavky na protokol:

- textový protokol nad transportním protokolem TCP nebo UDP
    - při shodě zadání v rámci cvičení (max. 2) bude každý student používat jiný
- bez šifrování
- využijte znalostí ze cvičení při návrhu (např. transparentnost při přenosu dat, apod.)
- když se nic neděje (žádný hráč nic nedělá), nic se neposílá
    - výjimkou může být občasná ping zpráva
- na každý požadavek přijde nějaká reakce (byť by šlo pouze o jednoznakové potvrzení, že se operace podařila)

Legenda:

- 🔴 červeně jsou označeny body, jejichž nesplnění automaticky vede k vrácení práce
- 🟡 oranžově jsou označeny body, které nemusí vést k vrácení práce (stále jsou ale povinné)
- 🔵 modře jsou označeny body, kvůli kterým práce již nebude vracena

Požadavky na aplikace:

- 🔴 aplikace jsou při odevzdání přeloženy standardním nástrojem pro automatický překlad (make, ant, maven, scons, ...; nikoliv bash skript, ani ručně gcc/javac, ani přes IDE)
    - pokud potřebujete nějakou knihovnu, verzi Javy, .. před prezentací si ji zajistěte a nainstalujte
- 🔴 je zakázáno využití jakékoliv knihovny pro síťovou komunikaci a serializace zpráv - to řeší váš kód sám pouze s BSD sockety (server) a nativní podporou ve standardní knihovně (klient; Java, C#, ..); není dovoleno použít ani C++2y networking rozhraní
- 🟡 kód aplikací je vhodně strukturovaný do modulů, resp. tříd
- 🟡 kód je dostatečně a rozumně dokumentovaný komentáři
- 🔴 aplikace (server i klient) jsou stabilní, nepadají na segfaultu (a jiných), všechny výjimky jsou ošetřené, aplikace se nezasekávají (např. deadlock)
- 🔴 počet hráčů ve hře je omezen pouze pravidly dané hry; vždy by však měla jít dohrát ve 2 lidech (abychom ji mohli testovat)
- 🟡 po vstupu se hráč dostane do "lobby" s místnostmi, hráč má možnost si vybrat místnost a vstoupit do ní (pokud nepřesahuje limit hráčů); případně je zařazen do fronty a čeká na naplnění herní místnosti
- 🔴 hra umožňuje zotavení po výpadku způsobené nečekaným ukončením klienta, krátkodobou nebo dlouhodobou síťovou nedostupností
    - dle pravidel hry se pak buď:
        - čeká na návrat hráče (hra se pozastaví)

        - nečeká na návrat hráče, hra pokračuje a po obnovení se hráč připojí do následujícího kola hry (ale hráči je stále po připojení obnoven stav)
        - nečeká na návrat hráče, hra pokračuje (ale hráči je stále po připojení obnoven stav)
    - krátkodobá nedostupnost nesmí nutit hráče k manuálnímu pokusu o připojení (klient vše provede automaticky)
    - dlouhodobá nedostupnost už by naopak měla (i s příslušnou zprávou hráči)
    - všichni hráči v dané hře musí vědět o výpadku protihráče (krátkodobém, dlouhodobém)
    - hráč, který je nedostupný dlouhodobě, je odebrán ze hry; hra pak může místnost ukončit (dle pravidel, většinou to ani jinak nejde) a vrátit aktivního protihráče zpět do lobby
- ● hráči jsou po skončení hry přesunuti zpět do "lobby"
- ● obě aplikace musí běžet bez nutnosti je restartovat (např. po několika odehraných hrách)
- ● obě aplikace ošetřují nevalidní síťové zprávy; protistranu odpojí při chybě
    - náhodná data nevyhovující protokolu (např. z /dev/urandom)
    - zprávy, které formátu protokolu vyhovují, ale obsahují očividně neplatná data (např. tah figurky na pole -1)
    - zprávy ve špatném stavu hry (např. tah, když hráč není ve hře/na tahu, apod.)
    - zprávy s nevalidními vstupy dle pravidel hry (např. šachy – diagonální tah věží)
    - ● aplikace můžou obsahovat počítadlo nevalidních zpráv a neodpojovat hned po první nevalidní zprávě, až po např. třech
- ● obě aplikace mají nějakou formu záznamu (log)
    - zaznamenávají se informace o stavech hráčů, her, popř. chybové hlášení, apod.

Server:

- ● server je schopen paralelně obsluhovat několik herních místností, aniž by se navzájem ovlivňovaly (jak ve smyslu hry, tak např. synchronizace)
- ● počet místností (limit) je nastavitelný při spouštění serveru, popř. v nějakém konfiguračním souboru
- ● celkový limit hráčů (dohromady ve hře a v lobby) je omezen; rovněž se dá nastavit při spuštění serveru nebo konfigurací
- ● stejně tak lze nastavit IP adresu a port, na které bude server naslouchat (parametr nebo konfigurační soubor; ne hardcoded)

Klient:

- ● klient implementuje grafické uživatelské rozhraní (Swing, JavaFX, Unity, popř. jiné dle možností zvoleného jazyka a prostředí) (ne konzole)
- ● klient umožní zadání adresy (IP nebo hostname) a portu pro připojení k serveru
- ● uživatelské rozhraní není závislé na odezvě protistrany - nezasekává se v průběhu např. připojení na server nebo odesílání zprávy/čekání na odpověď
- ● hráč a klient je jednoznačně identifikovaný přezdívkou (neřešíme kolize)
    - [nepovinné] chcete-li, můžete implementovat i jednoduchou registraci (přezdívka + heslo), aby se kolize vyřešily
- ● všechny uživatelské vstupy jsou ošetřeny na nevalidní hodnoty
    - totéž platí pro např. ošetření tahů ve hře (např. šachy, aby věž nemohla diagonálně, apod.)

- ● klient vždy ukazuje aktuální stav hry - aktuální hrací pole, přezdívky ostatních hráčů, kdo je na tahu, zda není nějaký hráč nedostupný, atp.
- ● klient viditelně informuje o nedostupnosti serveru - při startu hry, v lobby, ve hře
- ● klient viditelně informuje o nedostupnosti protihráče - ve hře

Dokumentace obsahuje:

- základní zkrácený popis hry, ve variantě, ve které jste se rozhodli ji implementovat
- popis protokolu dostatečný pro implementaci alternativního klienta/serveru:
    - formát zpráv
    - přenášené struktury, datové typy
    - význam přenášených dat a kódů
    - omezení vstupních hodnot a validaci dat (omezení na hodnotu, apod.)
    - návaznost zpráv, např. formou stavového diagramu
    - chybové stavy a jejich hlášení (kdy, co znamenají)
- popis implementace klienta a serveru (programátorská dokumentace)
    - dekompozice do modulů/tříd
    - rozvrstvení aplikace
    - použité knihovny, verze prostředí (Java), apod.
    - metoda paralelizace (select, vlákna, procesy)
- požadavky na překlad, spuštění a běh aplikace (verze Javy, gcc, ...)
- postup překladu
- závěr, zhodnocení dosažených výsledků

Průběžné odevzdání během semestru za bonusové body:

1) (cca 5. týden) popis protokolu - stavový diagram a přenášené zprávy (formát, apod.)
2) (cca 9. týden) kostra serveru, volitelně i klienta - připojení, elementární výměna zpráv, např. vylistování seznamu místností a možnost založit novou
3) (cca 12. týden) vyrovnání serveru/klienta s nevalidními stavy, řešení výpadků

Závěrečné odevzdání:

- minimálně je nutné získat alespoň 15 bodů, maximálně je možné získat až 30 bodů
- do termínu stanoveného cvičícím lze získat plný počet bodů (obvykle polovina ledna); poté je za každý den zpoždění (vč. sobot a nedělí) odečten 1 bod z celkového hodnocení práce
- odevzdání musí proběhnout nejpozději do konce ledna (mezní termín)
- student předvede funkčnost řešení na PC v laboratoři UC-326
- server je spuštěn na jednom PC s GNU/Linux, klient na jednom PC s GNU/Linux a druhém PC s MS Windows
- před odevzdáním si student připraví prostředí, aby předvádění mělo hladký průběh - ověří, zda se obě aplikace úspěšně přeloží na obou prostředích, zda je lze spustit a propojit
    - laboratoř je vám k dispozici, pokud v ní zrovna neprobíhá výuka, zkouška nebo jiná akce
- průběh odevzdání bude určitě zahrnovat (v režii studenta):
    - překlad klienta a serveru
    - spuštění s různými parametry
    - odehrání jedné celé hry bez výpadků (jejich simulace) a bez nevalidních dat
    - schopnost reagovat na výpadky (obě aplikace; dle zadání)

- schopnost vyrovnat se s nevalidními daty (obě aplikace; dle zadání)
- ověření náročnosti na systémové prostředky

Užitečné příkazy, tipy a triky:

- server, co naslouchá na 127.0.0.1:10000 a produkuje náhodná data:
  - cat /dev/urandom | nc -l 127.0.0.1 -p 10000
- klient, co se připojí na 127.0.0.1:10000 a produkuje náhodná data:
  - cat /dev/urandom | nc 127.0.0.1 10000
- simulace výpadku klienta/serveru (pouze vzdáleně):
  - stylem DROP (zahazuje pakety):
    - iptables -A INPUT -p tcp --dport 10000 -j DROP
  - stylem REJECT (odmítá pakety a odpovídá patřičnou ICMP zprávou):
    - iptables -A INPUT -p tcp --dport 10000 -j REJECT
  - odebrání pravidel - záměna -A za -D
- je vhodné (nikoliv povinné) zkusit, zda neuniká na serveru nějaká paměť (valgrind)
  - aby se zobrazila čísla řádku, kompilujte (gcc, clang) s přepínačem –g
  - opravením chyb, které valgrind vypíše, můžete výrazně snížit riziko „náhodných“ pádů a chyb v době odevzdání
- dodatečné verze Javy a knihoven neinstalujte do svého home – hrozí vyčerpání místa na vašem diskovém prostoru v AFS; místo toho použijte na Linuxu např. lokální složku /tmp
- pokud server padá na cílovém prostředí (Linux) a není jasné kde, přeložte rovněž s přepínačem -g a před jeho spuštěním použijte příkaz "ulimit -c unlimited" (popř. místo ‘unlimited‘ použijte nějaké rozumně velké množství paměti)
  - po pádu se vygeneruje soubor s názvem "core" - ten lze analyzovat nástrojem gdb, např. "gdb -c core muj-server" (za předpokladu, že zkompilovaná binárka serveru se jmenuje "muj-server")
  - v gdb můžete vypsat aktuální zanoření (zde v době pádu) příkazem "bt" ("backtrace")
  - případně můžete rovnou spouštět server uvnitř GDB ("gdb muj-server", příkaz "r")
  - podrobnější dokumentace nástroje GDB a jeho ovládání je například zde: http://sourceware.org/gdb/current/onlinedocs/gdb/