IF3230 Sistem Paralel dan Terdistribusi
Tugas Besar 1 Consensus Paxos - Werewolf

A. Background Story

Setelah bekerja seharian penuh, para member dari μ's, 765 Production, dan 346 Production melalui chatroom mereka memutuskan ingin memainkan sebuah permainan yang disebut Werewolf. Namun, saat mereka berdiskusi untuk berkumpul untuk memainkan game ini, mereka terkendala oleh jadwal mereka yang padat dan tak menentu sehingga sangat sulit bagi mereka yang sibuk itu untuk bermain bersama. Oleh karena itu salah seorang diantara produser mengusulkan untuk membuat sebuah aplikasi yang dapat dimainkan secara masif dan bebas oleh para member group tersebut. Pada kesempatan ini, peserta kuliah IF3230 mendapatkan kehormatan untuk membangun aplikasi tersebut. Aplikasi yang akan dibangun ini membutuhkan keahlian dari para peserta kuliah IF3230 yang memahami sistem terdistribusi terutama protokol paxos karena pada permainan itu akan banyak voting dan member yang bermain bisa saja mendapat panggilan mendadak dan permainan harus tetap berlanjut.

B. Spesifikasi Permainan

Aplikasi yang akan dibuat adalah aplikasi simulasi permainan *werewolf* yang telah disimplifikasi menjadi hanya dua peran, dengan minimal jumlah pemain adalah 6, yaitu 2 *werewolf* dan 4 penduduk desa. Peserta IF3230 diminta untuk membuat aplikasi dengan arsitektur *client-server*. Pemain dapat saling mengontak secara langsung melalui daftar IP pemain yang disediakan oleh *server*.

Alur dari permainan ini adalah sebagai berikut. Permainan dimulai pada siang hari tapi karena alasan tertentu urutan penjelasan dimulai dari malam hari.

1. Malam Hari
    a. Setiap malam, para *werewolf* akan mencapai kesepakatan target pembunuhan, dengan *werewolf* sendiri tidak termasuk sebagai target pembunuhan yang dapat dipilih. Pemilihan korban dilakukan berdasarkan hasil pemilihan mayoritas dari orang-orang dengan peran *werewolf*.
        i. Teknis penyampaian voting sama dengan pada siang hari.
        ii. *KPU* untuk malam hari adalah *KPU* yang terpilih pada siang hari (satu hari hanya satu *kpu*).
        iii. Bedanya dengan kasus siang hari, pemilihan korban dapat dilakukan berulang kali hingga tercapainya keputusan.
    b. Malam hari akan selesai ketika *werewolf* telah mencapai kesepakatan korban.

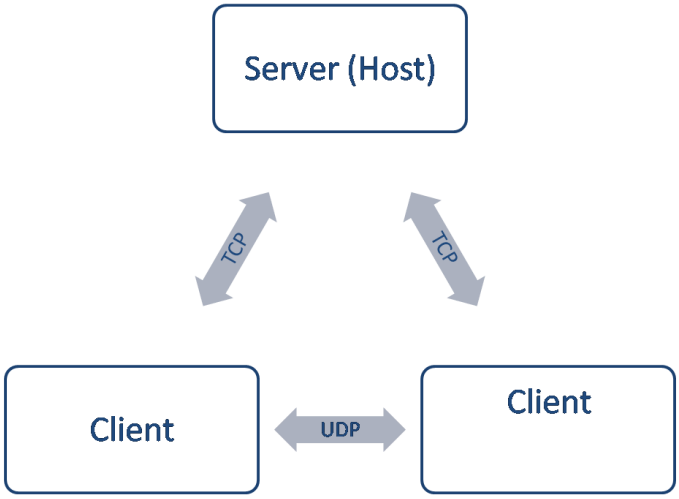
2. Siang Hari

a. Permainan dimulai dari siang hari, setiap pemain akan diberikan peran antara *werewolf* atau *civilian*. Pemain yang berperan sebagai *werewolf* akan diberikan informasi tambahan yaitu pemain lain yang memiliki peran sebagai *werewolf*.
b. Pemain non-*werewolf* tidak dapat melihat siapa *werewolf* selayaknya pada game asli.
c. Korban dari malam hari dinyatakan apabila ada (selalu ada kecuali hari pertama).
d. Dilakukan pemilihan *client* pengurus pemilihan suara (KPU) oleh *client* yang masih terlibat dalam permainan dengan *leader election* menggunakan algoritma *Paxos*. Proses pemilihan KPU ini bertujuan untuk memilih node yang akan menerima voting atas pemain yang diduga sebagai *werewolf*.
e. Setelah *KPU* didapatkan, dilakukan pengumpulan suara dari semua *client* yang ada (termasuk *KPU*). *KPU* akan meneruskan suara terbanyak kepada *server*.
    i. Untuk kasus tidak ada mayoritas (contoh: A mendapat 3 vote dan B mendapat 3 vote), maka dapat dilakukan pemilihan satu kali lagi.
    ii. Apabila pada pemilihan kedua juga tidak ada mayoritas, maka tidak ada yang dibunuh pada siang itu.
f. Hasil keputusan akan diberitahukan ke semua client melalui *server*.

Ada pula spesifikasi lainnya, yaitu sebagai berikut

1. Alamat *server* yang dimasukkan secara dinamik saat aplikasi berjalan (tidak di-*hardcode*).
2. Ronde permainan (hari dan waktu) dan peran masing-masing tetap ditampilkan pada layar masing-masing.
3. Antar *client-server* satu kelompok dengan kelompok lainnya dapat berkomunikasi tanpa masalah dengan protokol yang ditetapkan pada subbab C.
4. Permainan berakhir saat jumlah *werewolf* sama dengan jumlah penduduk, atau saat semua *werewolf* berhasil dibunuh.
5. Pemain yang telah dibunuh tetap mendapat informasi terbaru keadaan permainan.
6. Peran pemain yang telah terbunuh ditampilkan pada seluruh klien.
7. Untuk menguji berjalannya *Consensus Paxos*, terdapat modul kontrol yang berfungsi untuk mensimulasikan paket yang hilang saat pengiriman. Modul ini **wajib diimplementasikan** agar dapat mensimulasikan *unreliable connection* guna menguji kebenaran algoritma paxos yang diimplementasikan. Modul ini hanya digunakan dalam di bagian algoritma *paxos* (prepare dan accept). Protokol 5 dan 6 di bawah harus dilakukan dengan simulasi *unreliable connection*.
   Contoh kode dapat dilihat pada tautan berikut:
   https://www.dropbox.com/sh/jdth9vn1xzeun8d/AADJ8-w6Hl7buOrVQyNSgTZra
8. Bahasa pemrograman yang digunakan untuk pengerjaan tugas ini dibebaskan sesuai kehendak peserta.
9. User Interface dibebaskan kepada kreatifitas peserta, tetapi UI **tidak menjadi komponen penilaian utama** dari tugas ini sehingga utamakan fungsionalitas program dibandingkan tampilan dari client.

C. Protokol Komunikasi



Peserta praktikum diwajibkan untuk membuat dua program yaitu program server dan program client, adapun protokol komunikasi dari server ke client menggunakan protokol TCP, sedangkan protokol antar client menggunakan protokol UDP. Gambar di atas adalah ilustrasi protokol komunikasi yang harus diterapkan oleh seluruh peserta praktikum, diharapkan seluruh peserta untuk mengikuti protokol yang telah dibuat sehingga nantinya program antar kelompok dapat saling berkomunikasi.

Error dan Fail Response

Error adalah response yang diberikan ketika data yang diterima tidak sesuai dengan ketentuan sintaksis. Misalnya data yang diterima bukan objek JSON, objek JSON yang diterima tidak mengandung attribut yang dibutuhkan, dsb.

Fail adalah response yang diberikan ketika data yang diterima sesuai dengan ketentuan sintaksis, namun terdapat hal yang membuat request yang diberikan tidak dapat dieksekusi. Misalnya nilai yang diberikan tidak sesuai dengan protokol paxos, username yang diberikan ketika join sama dengan username pemain yang sudah join sebelumnya, dsb.

1. Join Game

Dilakukan di awal ketika client bergabung ke dalam permainan.

Flow	Client -> Server
Request example	{ "method": "join", "username": "sister", "udp_address": "192.168.1.3", "udp_port": 9999 }
OK Response example	{ "status": "ok", "player_id": 3 // **Dimulai dari 0** }
Fail Response example	{ "status": "fail", "description": "user exists" } **OR** { "status": "fail", "description": "please wait, game is currently running" }
Error Response example	{ "status": "error", "description": "wrong request" }

2. **Leave Game**

Dikirimkan oleh client kepada server ketika client keluar dari permainan.

Flow	Client -> Server
Request example	{ "method": "leave" }
OK Response example	{ "status": "ok", }
Fail Response example	{ "status": "fail", "description": <penjelasan>

	}
Error Response example	{ "status": "error", "description": "wrong request" }

3. **Ready Up**

Dikirimkan oleh client ke server ketika pemain siap untuk bermain.

Flow	**Client->Server**
Request example	{ "method": "ready" }
OK Response example	{ "status": "ok", "description": "waiting for other player to start" }
Fail Response example	
Error Response example	{ "status": "error", "description": "wrong request" }

4. **List Client**

Dikirimkan oleh client ke server ketika:

1. Client ingin mengetahui address dari UDP socket dari pemain lain
2. Client ingin mengetahui status/peran pemain.

Flow	**Client->Server**
Request example	{ "method": "client_address" }
OK Response example	{ "status": "ok", "clients" : [{ "player_id" : 0,

	"is_alive" : 1, "address" : "192.168.1.1", "port" : 9999, "username" : "sister" }, { "player_id" : 1, "is_alive" : 1, "address" : "192.168.1.2", "port" : 9999, "username" : "gaib" }, { "player_id" : 2, "is_alive" : 1, "address" : "192.168.1.3", "port" : 9999, "username" : "irk" }, { "player_id" : 3, "is_alive" : 0, "address" : "192.168.1.4", "port" : 9999, "username" : "basdat", "role" : "werewolf" }], "description": "list of clients retrieved" }
Fail Response example	{ "status": "fail", "description": <penjelasan> }
Error Response example	{ "status": "error", "description": <penjelasan> }

5. **Paxos Prepare Proposal (unreliable)**
   Dilakukan pada tahap konsesus paxos pada fase prepare. Yang bisa menjadi Proposer hanya dua client dengan player id terbesar (player ke n dan n-1). Acceptor disini adalah mayoritas client (boleh semua). <proposal-number> adalah nomor proposal yang

dimulai dari 1 dan selalu bertambah 1 setiap kali proposer membuat proposal baru. <proposal-number> bersifat lokal untuk sebuah proposer, setiap proposer memiliki counter-nya sendiri. `<previous_accepted_kpu_id>` adalah nilai yang sudah di accept oleh acceptor sebelumnya, jika tidak ada maka response tidak boleh mengandung atribut "previous_accepted". Perbandingan "proposal_id" dilakukan dengan membandingkan nilai yang pertama, jika nilai yang pertama sama nilai kedua yang akan dipakai dalam perbandingan. Contoh:

1. [a, b] > [c, d] jika a > c atau (a = c dan b > d)
2. [a, b] < [c, d] jika a < c atau (a = c dan b < d)
3. [a, b] = [c, d] jika a = c dan b = d

Flow	**Client (Proposer) -> Client(Acceptor)**
Request example	`{` `"method": "prepare_proposal",` `"proposal_id": [<proposal-number>, <player-id>]` `}`
OK Response example	`{` `"status": "ok",` `"description": "accepted",` `"previous_accepted": <previous_accepted_kpu_id>` `}`
Fail Response example	`{` `"status": "fail",` `"description": "rejected"` `}`
Error Response example	`{` `"status": "error",` `"description": <penjelasan>` `}`

6. Paxos Accept Proposal (unreliable)

Digunakan pada tahap konsesus paxos pada fase accept. Yang bisa menjadi Proposer hanya dua client dengan player id terbesar (player ke n dan n-1). Acceptor disini adalah mayoritas client (boleh semua). Atribut "kpu_id" disini adalah nilai yang di propose menurut algoritma paxos. Paxos memiliki aturan yang harus diikuti ketika proposer ingin memberikan nilai pada fase accept.

Flow	**Client (Proposer) -> Client(Acceptor)**
Request example	`{` `"method": "accept_proposal",`

	"proposal_id": [<proposal-number>, <player-id>], "kpu_id": <nilai sesuai protokol paxos>, }
OK Response example	{ "status": "ok", "description": "accepted", }
Fail Response example	{ "status": "fail", "description": "rejected" }
Error Response example	{ "status": "error", "description": <penjelasan> }

7. Client Accepted Proposal

Digunakan pada saat acceptor men-accept proposal. Learner akan mengetahui siapa yang terpilih ketika menerima request ini dari mayoritas acceptor.

Flow	**Client (Acceptor) -> Server(Learner)**
Request example	{ "method": "accepted_proposal", "kpu_id": <player-id>, "Description": "Kpu is selected" }
OK Response example	{ "status": "ok", "description": "" }
Fail Response example	{ "status": "fail", "description": <penjelasan> }
Error Response example	{ "status": "error", "description": <penjelasan> }

8. **Kill Werewolf Vote**

Dikirimkan oleh client (pemain) yang berperan sebagai werewolf ke KPU pada malam hari ketika melakukan voting siapa yang akan dibunuh oleh werewolf.

Flow	**Client -> KPU**
Request example	{ "method": "vote_werewolf", "player_id": 2 }
OK Response example	{ "status": "ok", "description": "" }
Fail Response example	{ "status": "fail", "description": "" }
Error Response example	{ "status": "error", "description": "" }

9. **Info Werewolf Killed**

Dikirimkan oleh KPU ke server pada malam hari ketika semua werewolf telah melakukan voting. "Vote_status" akan bernilai 1 ketika terdapat keputusan player yang dibunuh dan bernilai -1 ketika tidak ada keputusan.

Flow	**KPU->Server**
Request example	{ "method": "vote_result_werewolf", "vote_status": 1, "player_killed": 4, "vote_result": [[0, 1], [1, 2], …] } { "method": "vote_result", "vote_status": -1, "vote_result": [[0, 1], [1, 2], …] } // elemen vote_result

	`// [0,1]: player_id 0 divote oleh 1 pemain` `// [1,2]: player_id 1 divote oleh 2 pemain`
OK Response example	`{` `  "status": "ok",` `  "description": ""` `}`
Fail Response example	`{` `  "status": "fail",` `  "description": ""` `}`
Error Response example	`{` `  "status": "error",` `  "description": ""` `}`

10. **Kill Civilian Vote**

Dikirimkan oleh client (pemain) ke KPU ketika melakukan voting siapa yang akan dibunuh di siang hari.

Flow	**Client -> KPU**
Request example	`{` `  "method": "vote_civilian",` `  "player_id": 2` `}`
OK Response example	`{` `  "status": "ok",` `  "description": ""` `}`
Fail Response example	`{` `  "status": "fail",` `  "description": ""` `}`
Error Response example	`{` `  "status": "error",` `  "description": ""` `}`

11. **Info Civilian Killed**

Dikirimkan oleh KPU ke server pada siang hari ketika semua pemain telah melakukan voting. “vote_status” akan bernilai 1 ketika terdapat keputusan player yang dibunuh dan bernilai -1 ketika tidak ada keputusan.

Flow	**KPU->Server**
Request example	{ “method”: “vote_result_civilian”, “vote_status”: 1, “player_killed”: 4, “vote_result”: [[0, 1], [1, 2], …] } { “method”: “vote_result”, “vote_status”: -1, “vote_result”: [[0, 1], [1, 2], …] } **// elemen vote_result** **// [0,1]: player_id 0 divote oleh 1 pemain** **// [1,2]: player_id 1 divote oleh 2 pemain**
OK Response example	{ “status”: “ok”, “description”: “” }
Fail Response example	{ “status”: “fail”, “description”: “” }
Error Response example	{ “status”: “error”, “description”: “” }

12. **Start Game**

Dikirimkan oleh server ke client ketika permainan dimulai.

Flow	**Server -> Client**
Request example	{ “method”: “start”, “time”: “day”, “role”: “werewolf”, “friend”: [“ahmad”, “dariel”],

	"description": "game is started", } // Role available: "werewolf" and "civilian" // friend is given if the role is werewolf, otherwise it is empty
OK Response example	{ "status": "ok" }
Fail Response example	{ "status": "fail", "description": "" }
Error Response example	{ "status": "error", "description": "" }

13. **Change Phase**

Dikirimkan oleh server ke client ketika perubahan waktu. Baik dari siang ke malam maupun dari malam ke siang.

Flow	**Server -> Client**
Request example	{ "method": "change_phase", "time": "day", "days": 3, "description" : "" }
OK Response example	{ "status": "ok", }
Fail Response example	{ "status": "fail", "description": "" }
Error Response example	{ "status": "error", "description": "" }

14. **Vote**

Dikirimkan oleh server ketika akan dilakukan voting pemain yang akan dibunuh. Voting disini bukan pemilihan KPU. Jika phase = “day”, gunakanlah protokol 10. Jika phase = “night”, gunakan lah protokol 8. Voting ulang juga dilakukan dengan menggunakan protokol ini.

Flow	**Server -> Client**
Request example	{ “method”: “vote_now”, “phase”: “day” } // day: “day” | “night”
OK Response example	{ “status”: “ok”, }
Fail Response example	{ “status”: “fail”, “description”: “” }
Error Response example	{ “status”: “error”, “description”: “” }

15. **Game Over**

Dikirimkan oleh server ketika permainan berakhir.

Flow	**Server -> Client**
Request example	{ “method”: “game_over”, “winner”: “werewolf”, “description”: “” }
OK Response example	{ “status”: “ok”, }
Fail Response example	{ “status”: “fail”, “description”: “”

	}
Error Response example	{ "status": "error", "description": "" }

D. Pengerjaan & Pengumpulan

Tugas dikerjakan secara berkelompok, dimana terdapat maksimal 3 orang untuk tiap kelompok (berada pada kelas yang sama). Deadline untuk pengumpulan Tugas Besar IF3230 adalah 1 Mei 2016, 21.00. Mekanisme pengumpulan serta jadwal untuk demo tugas besar akan diumumkan kemudian melalui grup milis. Adapun segala kecurangan akan mendapatkan konsekuensi. Bila ada pertanyaan yang kurang jelas mengenai spesifikasi soal, peserta dapat menggunakan milis sebagai sarana diskusi dan tanya jawab.

Selamat mengerjakan!

Link menuju referensi mengenai paxos:

1. https://www.quora.com/Distributed-Systems-What-is-a-simple-explanation-of-the-Paxos-algorithm/answer/Vineet-Gupta?srid=Cehd
2. http://the-paper-trail.org/blog/consensus-protocols-paxos/
3. https://raft.github.io/ (Have some similarities with paxos, good visualization to understand the protocol)