System Programming Presentation

7_X Algorithm

by baemincheon

index

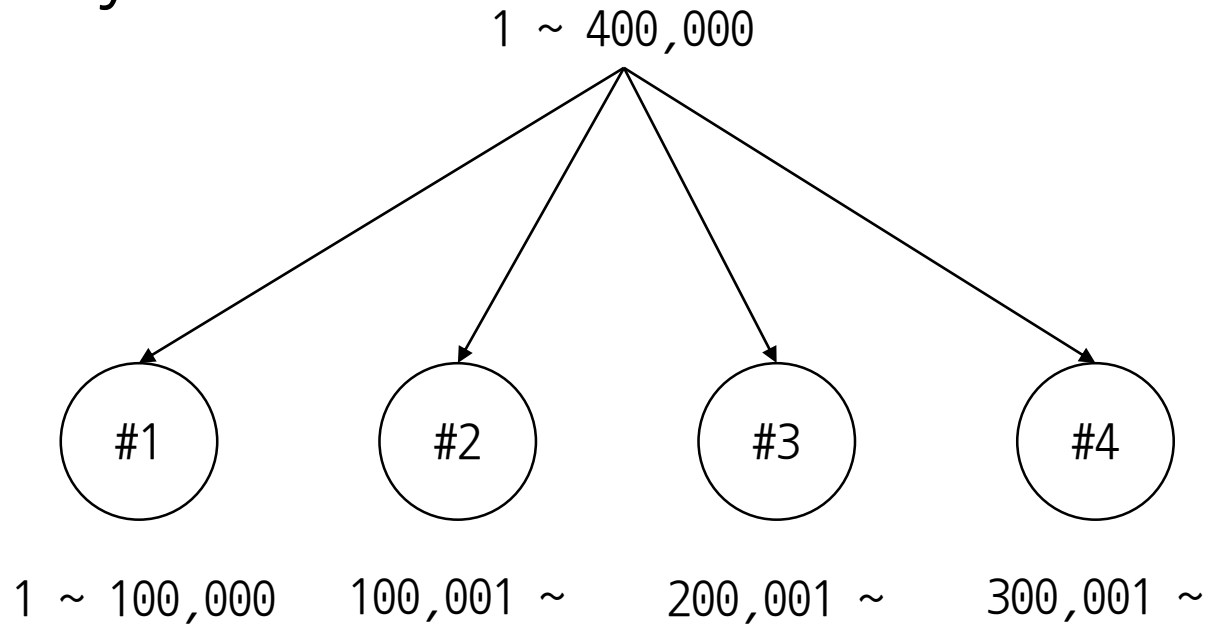
1. Try #1
    1) concept
    2) implementation
    3) limitation
2. Try #2
    1) concept
    2) implementation
    3) result
3. QnA

for getting code, visit github : https://github.com/BaeMinCheon/linux-c-multicore-prime-numbers

1. Try #1

1. Try #1

1) concept



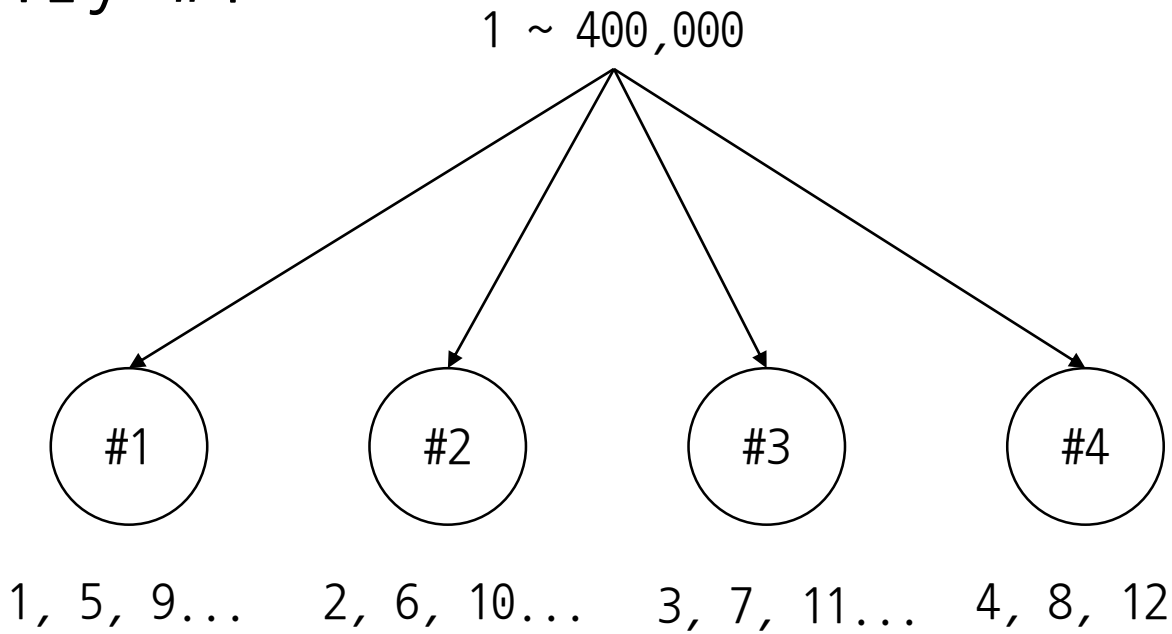
작업범위를 어떻게 나눠야할까 ?

```
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$ ./a.out 1 400000 4
[ parameter list ]
- range : (1 ~ 400000)
- number of process : 4

PID 267 has found 9592 prime number(s)
elapsed time : 1703 milli-second(s)
PID 268 has found 8392 prime number(s)
elapsed time : 4128 milli-second(s)
PID 269 has found 8013 prime number(s)
elapsed time : 6249 milli-second(s)
PID 270 has found 7863 prime number(s)
elapsed time : 8348 milli-second(s)
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$
```

순서대로 등분할 경우 뒤쪽 프로세스에게 작업량이 집중되는군

1. Try #1



프로세스 개수만큼을 오프셋으로 준다면 ?

너무 쉽게 풀리는 게 아닌가 ?

1. Try #1


```
int IsPrimeNumber(int _n)
{
	if(_n < 2)
	{
		return 0;
	}
	else
	{
		int retVal = 1;

		for(int i = 2; i < _n; ++i)
		{
			if((_n % i) == 0)
			{
				retVal = 0;
				break;
			}
		}

		return retVal;
	}
}

int GetNumberOfPrimeNumber(int _s, int _e)
{
	int retVal = 0;

	for(int i = _s; i <= _e; ++i)
	{
		if(IsPrimeNumber(i))
		{
			++retVal;
		}
	}

	return retVal;
}
```

순서대로 등분시

```
pid_t myPID = -1;
TimerInit(number_process);
int gap = (int)(range_end / number_process);
for(int i = 0; i < number_process; ++i)
{
	myPID = fork();

	if(myPID == 0)
	{
		TimerStart(i);
		printf("PID %d has found %d prime number(s) \n", (int)getpid(),
		    GetNumberOfPrimeNumber(range_start + i * gap, range_start + (i + 1) * gap));
		TimerEnd(i);
		TimerPrint(i);
		return 0;
	}
	else
	{
		continue;
	}
}
```

1. Try #1


```
int IsPrimeNumber(int _n)
{
	if(_n < 2)
	{
		return 0;
	}
	else
	{
		int retVal = 1;

		for(int i = 2; i < _n; ++i)
		{
			if((_n % i) == 0)
			{
				retVal = 0;
				break;
			}
		}

		return retVal;
	}
}

int GetNumberOfPrimeNumber(int _s, int _e, int _offset)
{
	int retVal = 0;

	for(int i = _s; i <= _e; i += _offset)
	{
		if(IsPrimeNumber(i))
		{
			++retVal;
		}
	}

	return retVal;
}
```

프로세스 개수만큼 오프셋 주기

```
pid_t myPID = -1;
TimerInit(number_process);
for(int i = 0; i < number_process; ++i)
{
	myPID = fork();

	if(myPID == 0)
	{
		TimerStart(i);
		printf("PID %d has found %d prime number(s) \n", (int)getpid(),
		    GetNumberOfPrimeNumber(range_start + i, range_end, number_process));
		TimerEnd(i);
		TimerPrint(i);
		return 0;
	}
	else
	{
		continue;
	}
}
```

프로세스 개수만큼 오프셋 주기의 결과

```
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$ ./a.out 1 400000 4
[ parameter list ]
- range : (1 ~ 400000)
- number of process : 4

PID 279 has found 1 prime number(s)
elapsed time : 1 milli-second(s)
PID 281 has found 0 prime number(s)
elapsed time : 1 milli-second(s)
PID 280 has found 16959 prime number(s)
elapsed time : 9917 milli-second(s)
PID 278 has found 16900 prime number(s)
elapsed time : 9989 milli-second(s)
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$
```

아니 멀티코어 활용도는 더 떨어졌는데 ??

동일한 오프셋이기 때문에 특정 프로세스들의 작업량이 현저히 감소

짝수 + 짝수 = 짝수 ...를 잊지말자

1. Try #1

홀수와 짝수의 분배가 중요하다 !

- **프로세스 개수 = 2**
  - 프로세스 #1 : 1, 3, 5, 7...
  - 프로세스 #2 : 2, 4, 6, 8...
- **프로세수 개수 = 3**
  - 프로세스 #1 : 1, 4, 7, 10...
  - 프로세스 #2 : 2, 5, 8, 11...
  - 프로세스 #3 : 3, 6, 9, 12...

죄다 짝수라서 소수 판별이 금방 끝나버린다

얼핏 보면 균형잡힌 것 같지만 함정이 있다
프로세스 #3이 3의 배수만 검사하기 때문이다

```
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$ ./a.out 1 400000 3
[ parameter list ]
- range : (1 ~ 400000)
- number of process : 3

PID 32 has found 1 prime number(s)
elapsed time : 2 milli-second(s)
PID 30 has found 16892 prime number(s)
elapsed time : 9622 milli-second(s)
PID 31 has found 16967 prime number(s)
elapsed time : 9675 milli-second(s)
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$
```

홀수와 짝수의 분배가 중요하다 !

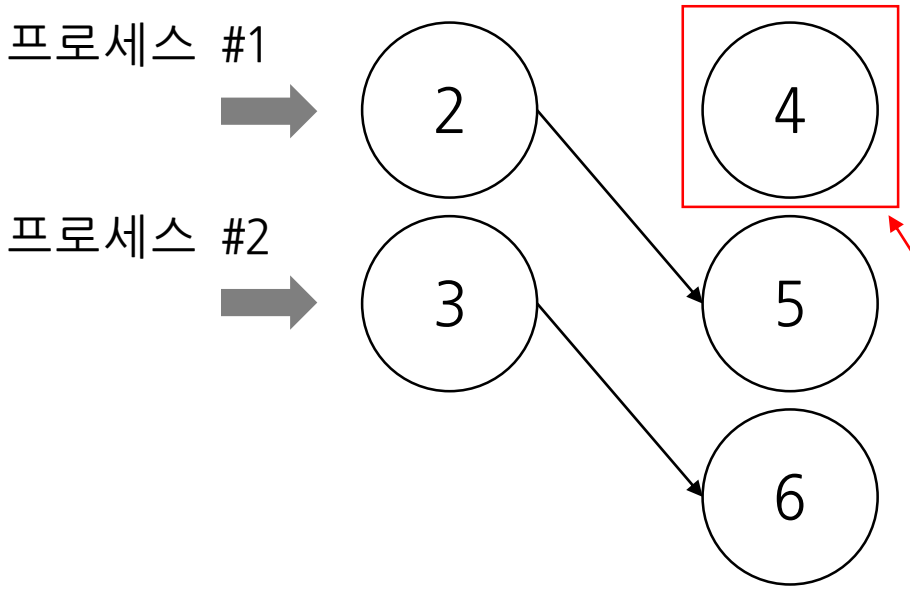
프로세스가 홀수만 처리하지않도록, 짝수로 바꿔줘야할 것이고...
짝수만 처리하지않도록, 홀수로 바꿔줘야할 것이다 !

홀수에서 짝수로, 짝수에서 홀수로 만드는 가장 쉬운 방법은 ?

홀수 + 1 = 짝수, 짝수 + 1 = 홀수 임을 명심하자

2. Try #2

2. Try #2



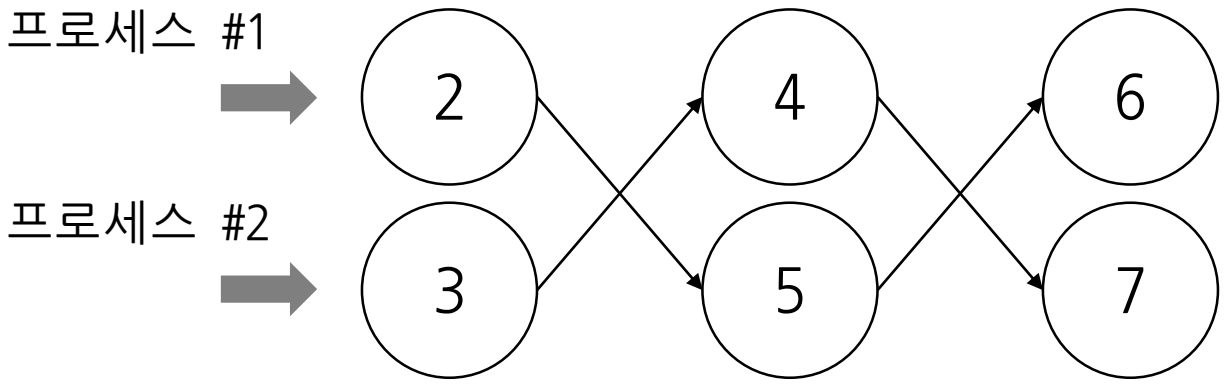
오프셋 = 프로세스 개수 + 1 = 3

이렇게 된 이상 오프셋을 프로세스 개수보다 크게 잡고...

이로 인해 발생하는 잉여 숫자를 이용해볼까 ?
홀수와 짝수를 분배하는 데에 활용해보자

2. Try #2

1) concept



잉여 숫자 분배를 위해 오프셋을 주기적으로 바꾸자

기본 오프셋 = 프로세스 개수 + 1 = 3

특수 오프셋 = 1

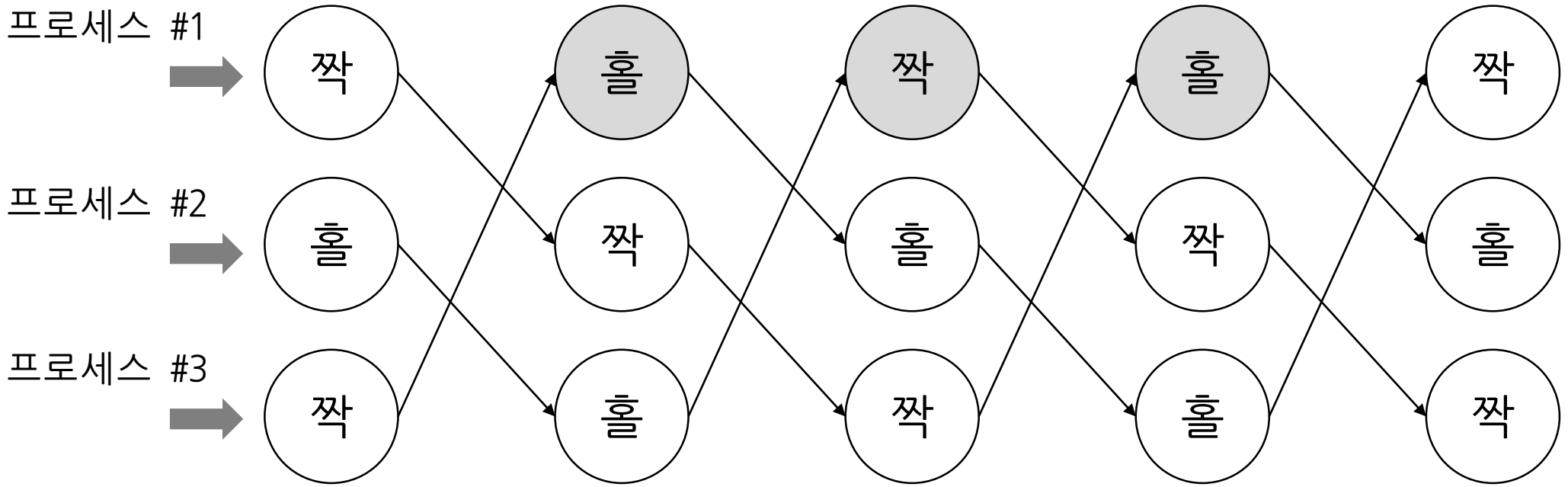
프로세스 개수보다 1 크다면 프로세스 하나의 오프셋만 변경하면 되겠군

2. Try #2

1) concept

- 기본 오프셋 : 3 + 1 = 짝수

- 탐색 범위 : 짝수부터

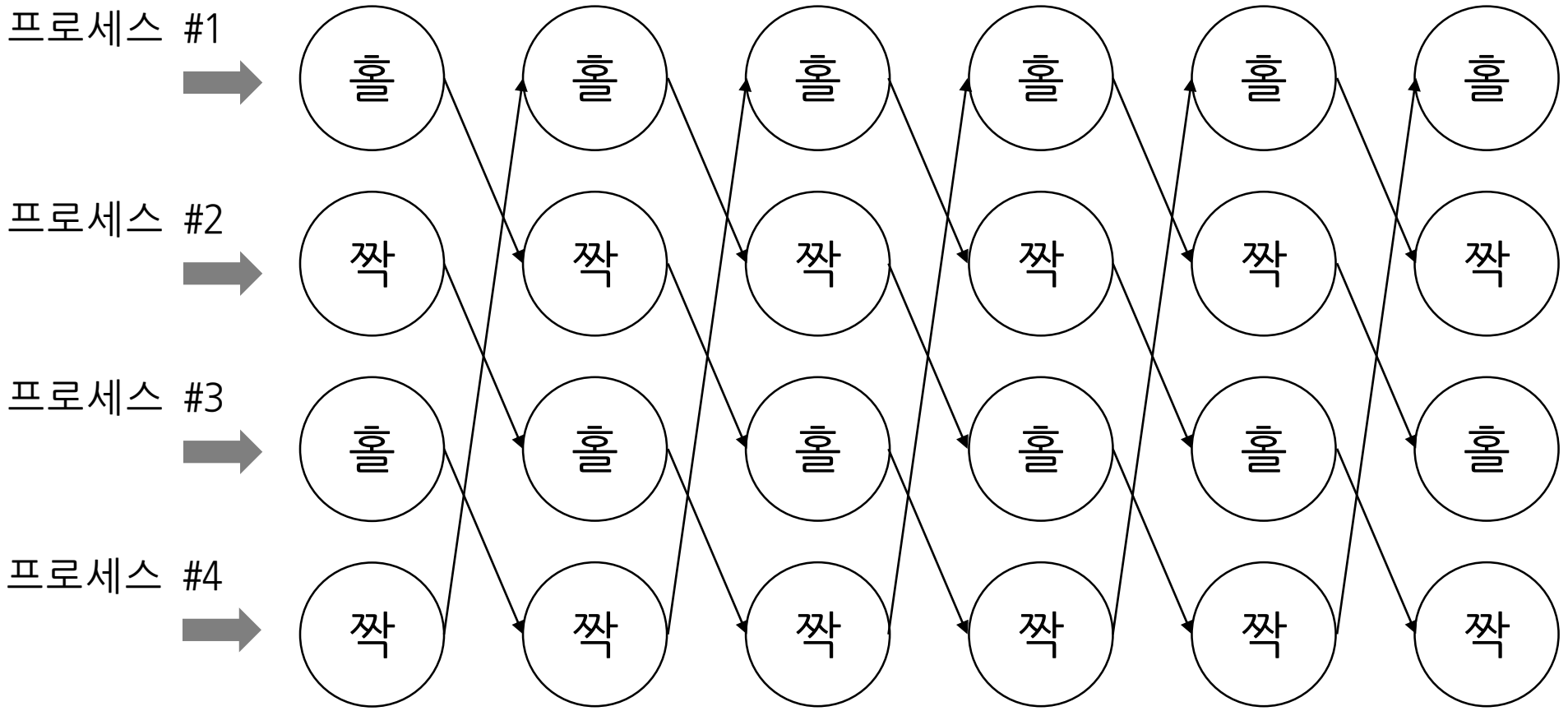


홀수/짝수를 유지하던 프로세스도 짝수/홀수로 변경

2. Try #2

- 기본 오프셋 : 4 + 1 = 홀수

- 탐색 범위 : 홀수부터



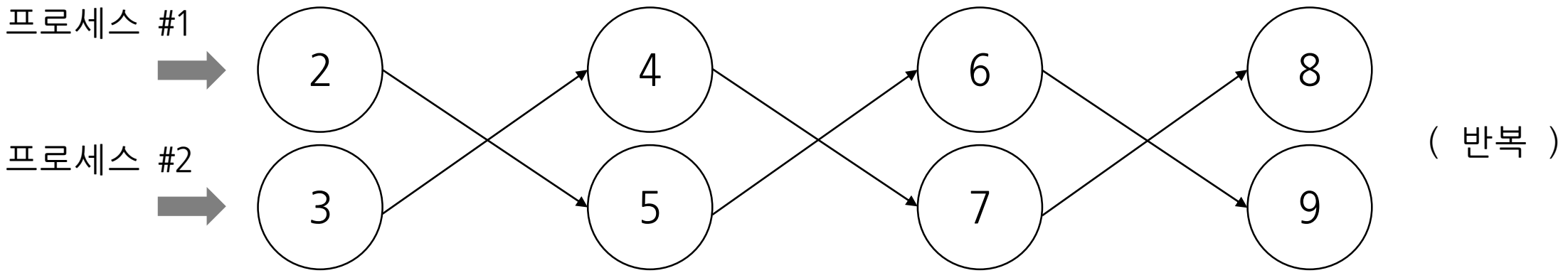
(과제에 사용된) 동적 오프셋 방법의 조건

- **초기값**
  - 기본 오프셋 = 프로세스 개수 + 1
  - 특수 오프셋 = 1
- **조건**
  - 각 프로세스는 “ 프로세스 개수 – 1 ” 번만큼의 소수판별 작업을 수행한 경우, 특수 오프셋을 한 번 사용한다. 그 이후 다시 기본 오프셋으로 돌아온다.
  - 단, 초기 작업 시작시에만 프로세스 순서에 따라 오프셋 변환 카운터값을 달리 한다. 마지막 프로세스부터 역순으로 특수 오프셋을 사용하도록 하기 위함이다.

결과적으로, 홀수와 짝수를 적절하게 분배

2. Try #2

ex) ./7_X.out 2 1000 2

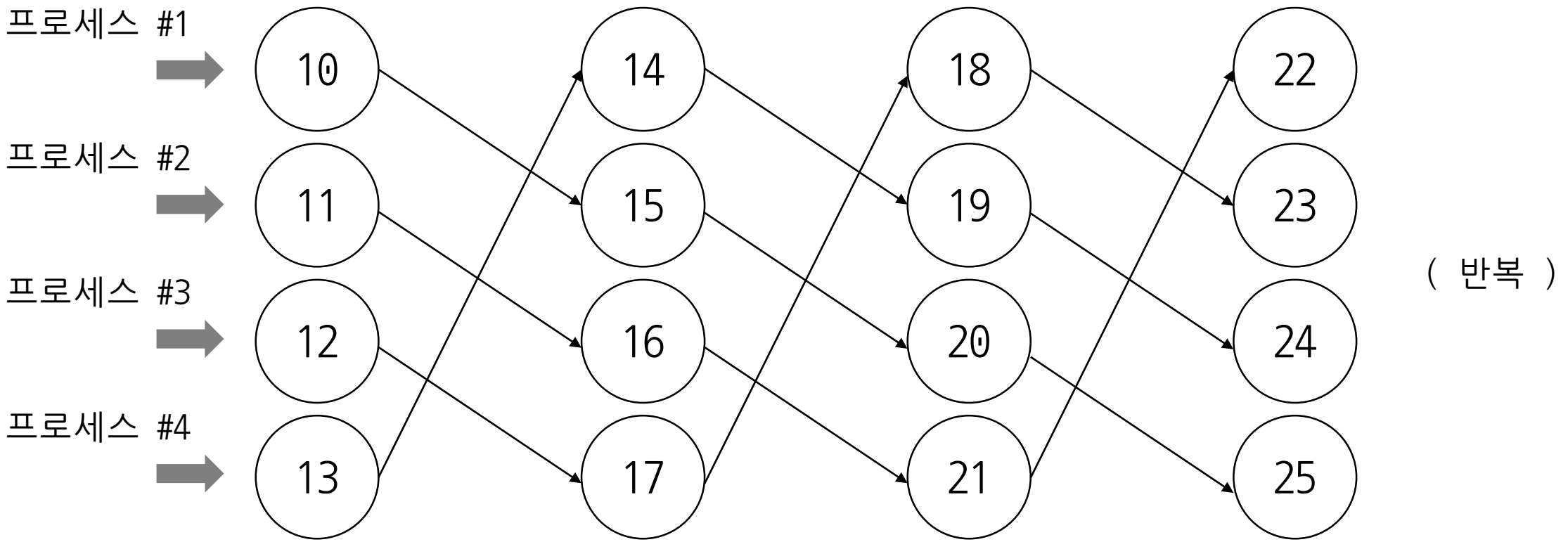


기본 오프셋 = 프로세스 개수 + 1 = 3

특수 오프셋 = 1

2. Try #2

ex) ./7_X.out 10 10000 4



기본 오프셋 = 프로세스 개수 + 1 = 5

특수 오프셋 = 1

2. Try #2


```
int IsPrimeNumber(int _n)
{
	if(_n < 2)
	{
		return 0;
	}
	else
	{
		int retVal = 1;

		for(int i = 2; i < _n; ++i)
		{
			if((_n % i) == 0)
			{
				retVal = 0;
				break;
			}
		}

		return retVal;
	}
}

int GetNumberOfPrimeNumber(int _s, int _e, int _rank, int _np)
{
	int retVal = 0;

	for(int i = _s; i <= _e;)
	{
		if(IsPrimeNumber(i))
		{
			++retVal;
		}
		if(_rank == (_np - 1))
		{
			_rank = 0;
			i += 1;
		}
		else
		{
			++_rank;
			i += _np + 1;
		}
	}

	return retVal;
}
```

동적 오프셋 방법

```
pid_t myPID = -1;
TimerInit(number_process);
for(int i = 0; i < number_process; ++i)
{
	myPID = fork();

	if(myPID == 0)
	{
		TimerStart(i);
		printf("PID %d has found %d prime number(s) \n", (int)getpid(),
		    GetNumberOfPrimeNumber(range_start + i, range_end, i, number_process));
		TimerEnd(i);
		TimerPrint(i);
		return 0;
	}
	else
	{
		continue;
	}
}
```

2. Try #2

3) result

$$\frac{MaxRunTime}{MinRunTime} = \frac{5857}{5599} = 1.046079 \cdots < 1.3$$

```
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$ ./7_X.out 1 400000 4
[ parameter list ]
- range : (1 ~ 400000)
- number of process : 4

root pid : 21

PID 22 has found 8449 prime number(s)
elapsed time : 5599 milli-second(s)
PID 23 has found 8468 prime number(s)
elapsed time : 5656 milli-second(s)
PID 24 has found 8424 prime number(s)
elapsed time : 5717 milli-second(s)
PID 25 has found 8519 prime number(s)
elapsed time : 5857 milli-second(s)
thanang@LAPTOP-THANANG:~/assignment/Lab07/7-X$
```

3. QnA

THANK YOU
FOR
YOUR ATTENTION